Probabilistic Model Checking on HPC Systems for the Performance Analysis of Mobile Networks*

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

Tamas Berczes and Janos Sztrik
Department of Informatics Systems and Networks, Faculty of Informatics
University of Debrecen, Debrecen, Hungary
sztrik.janos@inf.unideb.hu, berczes.tamas@inf.unideb.hu

September 19, 2013

Abstract

We report on the use of HPC resources for the performance analysis of the mobile cellular network model described in "A New Finite-Source Queueing Model for Mobile Cellular Networks Applying Spectrum Renting" by Tien v. Do et al. That paper proposed a new finite-source retrial queueing model with spectrum renting that was analyzed with the MOSEL-2 tool. Our results show how this model can be also appropriately described and analyzed with the probabilistic model checker PRISM, although at some cost considering the formulation of the model; in particular, we are able to accurately reproduce most of the analytical results presented in that paper and thus validate the previously presented results. However, we also outline some discrepancies with may hint to deficiencies of the original analysis. Moreover, by applying a parallel computing framework developed for this purpose, we are able to considerably speed up studies performed with the PRISM tool.

*Supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. This project has been supported by the European Union, co-financed by the European Social Fund.

Contents

1. Introduction

We report in this paper on the application of high performance computing (HPC) resources for the performance analysis of mobile networks. We use the mobile cellular network system that was introduced in [2] where a number of sources (cell phone subscribers) compete for access to a number of servers (channels). Sources produce requests at rate λ; a free server processes these requests at rate μ. However, the number of available channels varies: if it gets to small, the cell phone operator may rent additional frequency blocks from another operator, partition these blocks into channels, and add these new channels to its own pool. If sufficiently many channels have become free again, the rented blocks may be released.

In [2], this model was originally analyzed with the help of the performance modeling tool MOSEL-2 [1] which is however not supported any more. Our own results are derived with the help of the probabilistic model checker PRISM [3, 4] which is actively developed and has been used for numerous purposes, among them the performance analysis of computing systems. In [5], we have developed an initial version of the model in PRISM which was subsequently refined and corrected in [6]. Furthermore, we have in [6] described a parallel computing framework that we applied to analyze the PRISM model with the use of HPC resources, i.e., we have speed up the analysis of our model by running experiments on a massively parallel non-uniform memory architecture (NUMA).

However, in [5, 6] only a small number of experiments were performed, some of which derived different results than were originally reported in [2]. This paper complements our work by presenting all experiments that were also described in [2] and illustrating for the whole set of experiments the speedup that can be achieved by their execution in our parallel computing framework.

The remainder of this paper is structured as follows: to make this paper self-reliant, we summarize in Section 2 the previously introduced model and the parallel execution framework. In Section 3, we present our new results and contrast them to those reported in [2]. Section 4 presents our conclusions and open issues for further work.

This research was realized in the frames of TÁMOP 4.2.4. A/2-11-1-2012-0001 “National Excellence Program – Elaborating and operating an inland student and researcher personal support system”. The project was subsidized by the European Union and co-financed by the European Social Fund.

2. The Model

Appendix B presents the PRISM model that was introduced in [5]: it applies the concept of spectrum renting for mobile cellular networks introduced in [2]. However, that paper also shows results for a corresponding model (that is not described in detail there) without spectrum renting. In order to repeat the corresponding experiments, we give in Appendix A a version of our PRISM model from which spectrum renting has been stripped but which is otherwise identical.

The experiments of this paper were performed with the execution script listed in Appendix C; it applies the parallel execution framework (command `parallel`) introduced in [5]. The experiments were performed on an SGI Altix UltraViolet 1000 supercomputer installed at the

P	t_p (s)			T_p (s)	S_p
1	3813	3788	3792	3798	1.0
2	1949	1950	1944	1948	1.9
4	1028	1025	1018	1024	3.7
8	559	561	560	560	6.8
16	334	328	329	330	11.5
32	252	252	244	249	15.3

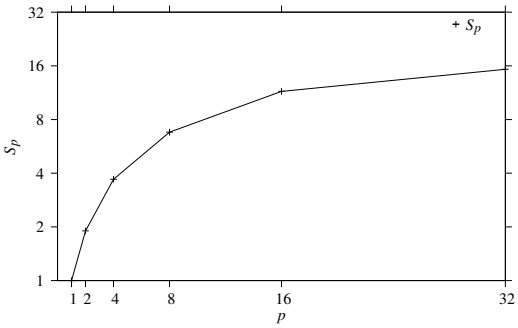

Figure 1: Execution Times and Speedups

Johannes Kepler University Linz. This machine is equipped with 256 Intel Xeon E78837 processors with 8 cores each which are distributed among 128 nodes with 2 processors (i.e. 16 cores) each; the system thus supports computations with up to 2048 cores. Access to this machine is possible via interactive login; by default every user may execute threads on 4 processors with 32 cores and 256 GB memory. Since PRISM is implemented in Java, we applied the execution script `prism-java` described in [5] which calls `java` with memory allocation and optimization optimized for execution on a NUMA system.

3. The Analysis

With the help of our parallel execution framework, we have performed for our PRISM model all the experiments that were also reported in [2]; the results are depicted in Figures 2 to 10 (with references to the corresponding figures presented in [2]). The experiments shown in Figure 2 (corresponding to Figure 2 in [2]) are performed in the model without spectrum renting (see Appendix A); all other ones are performed in the model with spectrum renting (see Appendix B); in the later case appropriate variants of the model were used as required by the different sets of parameters with varying respectively fixed values.

From the 29 experiments (comprising in total 920 PRISM runs to produce the various data points of each experiment), 25 show results that are virtually identical to those presented in [2]. This correspondence strongly validates both the original MOSEL-2 model and our PRISM model. However, there are also four notable discrepancies:

- As already stated in [6], in Figure 3 (corresponding to Figure 3 of [2]) the two bottom diagrams show in our model (especially for low traffic intensity ρ_0) a lower mean number of rented blocks mB and a lower mean number of busy channels mC than originally reported (while the overall shape of the curves are similar).

- Figures 9 and 10 (corresponding to Figures 9 and 10 of [2]) reporting on the impact of retrials on the average profit rate (APR) and on the average number of busy channels (mC) show for the first parameter set $\rho_0 = 0.4, p_{io} = 0.8$ the same results as originally reported;

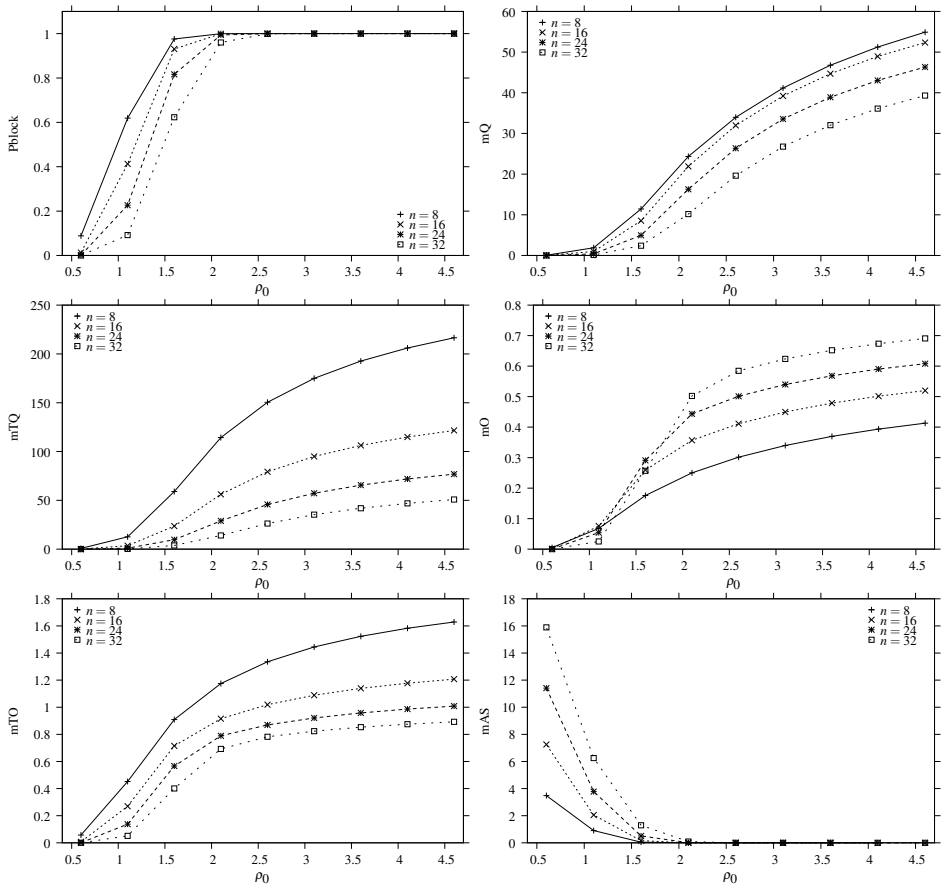

Figure 2: Performance Measures When There Is No Renting (cf. Fig. 2 from [2])

however for the two other parameter sets our experiments report significantly lower figures, i.e., the three lines are much farther apart than in [2].

Since in all other cases the results are identical to the other reports and we have both carefully checked our model and the deviating experiments, the possibility remains that the errors are in the originally reported experiments. We have therefore asked one of the authors of [2] to re-check these experiments.

As for the time needed for executing the analysis, Figure 1 lists the times (in seconds) for performing all the 920 PRISM runs illustrated in Figures 2–10 with P processes, $1 \leq P \leq 32$ (the maximum number of processor cores available to us for this experiment). The analysis was performed five times from which we have excluded the fastest and the slowest run. This leads to three values for the execution time t_p with average execution time T_p; the speedup for this average is reported as S_p. We see that significant speedups up to a maximum of 15.3 can be achieved; for $P = 32$, improvements are still visible but will (for this experiment) certainly become marginal for a larger number of processes.

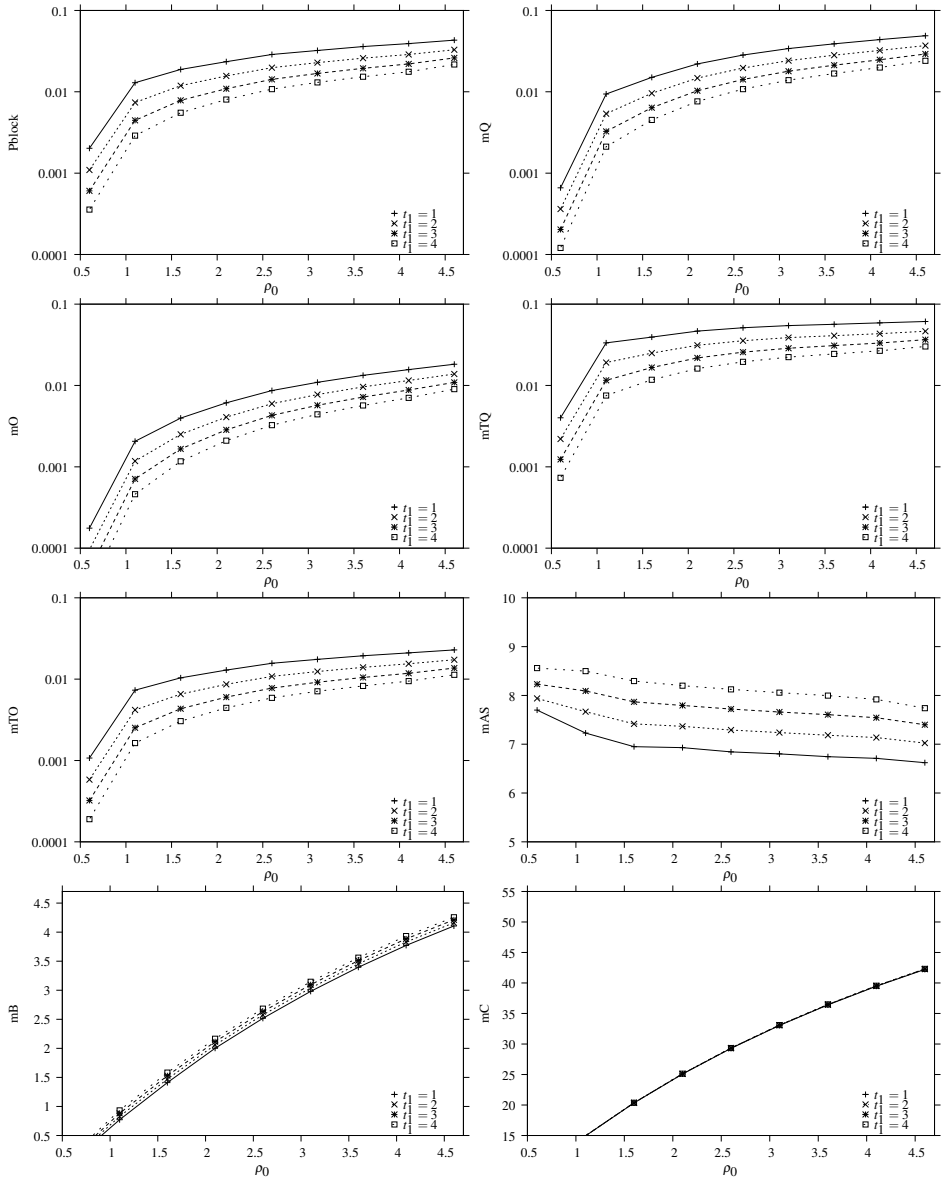

Figure 3: Performance Measures for $t_2 = 6$ (cf. Fig. 3 from [2])

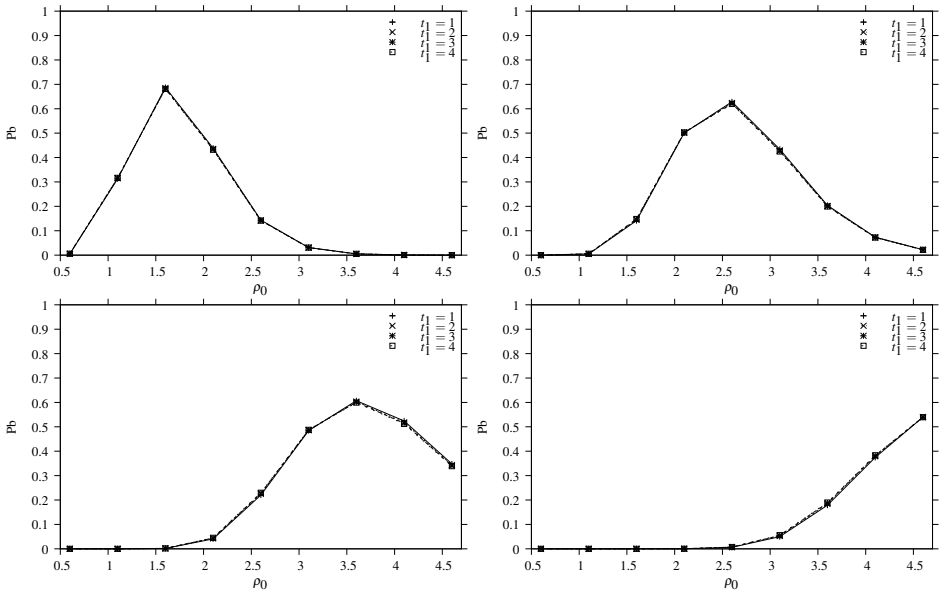

Figure 4: Further Performance Measures for $t_2 = 6$ (cf. Fig. 4 from [2])

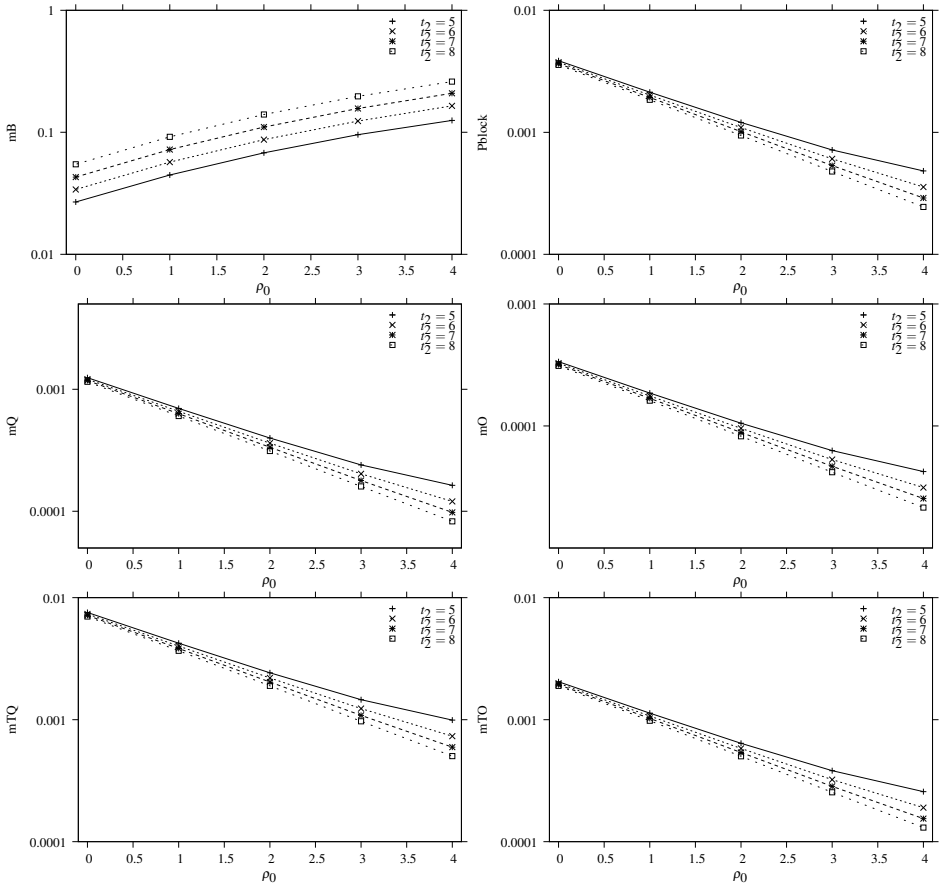

Figure 5: Performance Measures for $\rho_0 = 0.6$ (cf. Fig. 5 from [2])

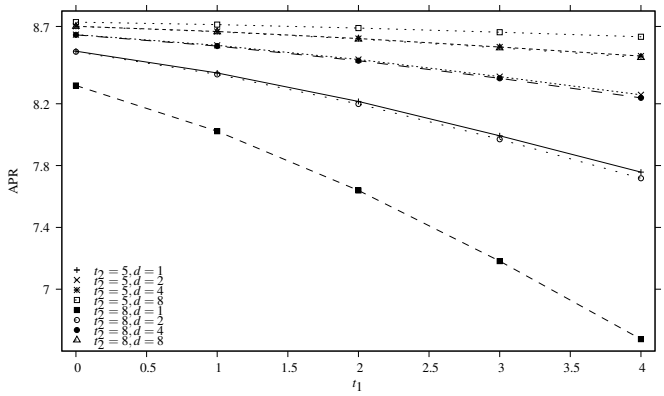

Figure 6: *APR* vs. t_1 and d for $\rho_0 = 0.6$ (cf. Fig. 6 from [2])

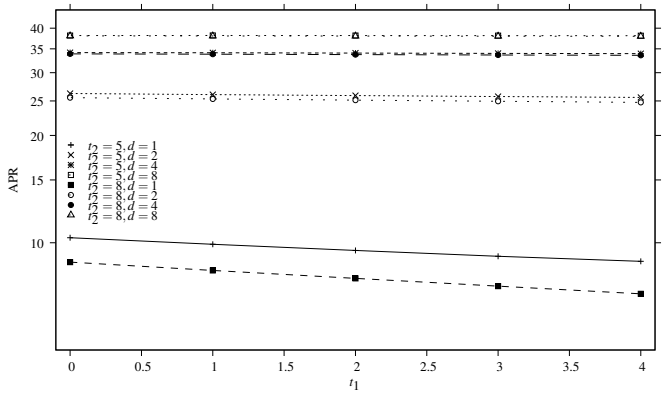

Figure 7: *APR* vs. t_1 and d for $\rho_0 = 4.6$ (cf. Fig. 7 from [2])

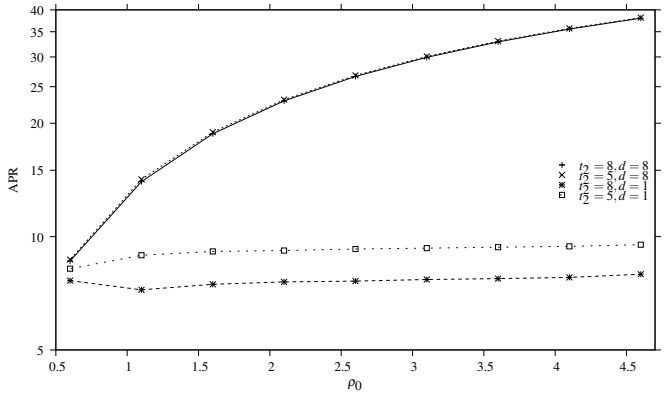

Figure 8: *APR* vs. ρ_0 and d (cf. Fig. 8 from [2])

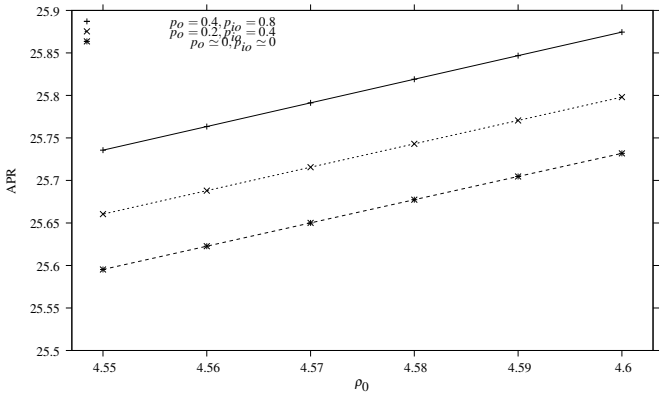

Figure 9: Impact of retrials on *APR* (cf. Fig. 9 from [2])

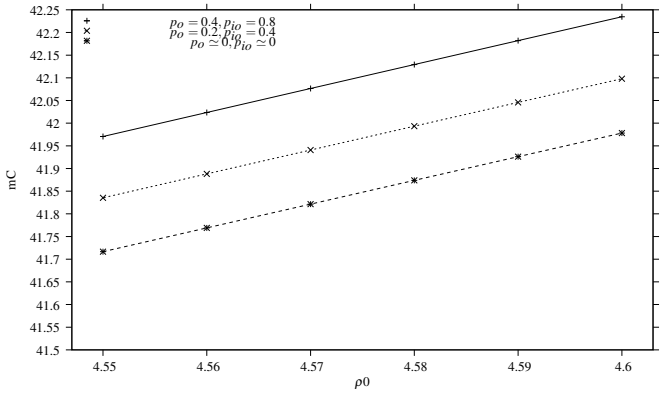

Figure 10: Impact of retrials on the average number of busy channels (cf. Fig. 9 from [2])

4. Conclusions

We have shown in this report how the PRISM analysis of a non-trivial mobile cellular network can be efficiently performed on a modern high performance computing system and how by this analysis the results performed with the older (and not any more supported) MOSEL-2 tool can be essentially validated. However, as already reported in [6], a crucial difference between MOSEL-2 and PRISM (the existence respectively lack of zero-time/infinite-rate transitions) makes the PRISM model somewhat more unhandy than originally thought; more efforts are needed in in PRISM to express the desired models in an economical way.

Furthermore, while most of the originally reported results (25 of 29 experiments) could be validated, still some discrepancies (in 4 experiments) have to be resolved. While the error may well be in the PRISM model or its analysis, it might as well be true that there are errors in the originally reported results (we have asked one author of the original paper for a re-examination of these experiments). This demonstrates that the performance analysis of computing systems

by analyzing system models alone cannot give full confidence in the correctness of the results: further verification (by comparison against measurements of the actual system) or validation (by comparison with the analysis of another model by another tool) is highly recommended.

References

[1] K. Begain, G. Bolch, and Herold H. *Practical Performance Modeling Application of the MOSEL Language*. Kluwer Academic Publisher, 2012.

[2] Tien v. Do, Patrick Wüchner, Tamas Berczes, Janos Sztrik, and Hermann de Meer. "A New Finite-Source Queueing Model for Mobile Cellular Networks Applying Spectrum Renting". In: *Asia-Pacific Journal of Operational Research* (2013). To appear.

[3] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591.

[4] David A. Parker, ed. *PRISM — Probabilistic Symbolic Model Checker*. http://www.prismmodelchecker.org. Department of Computer Science, University of Oxford, UK. 2013.

[5] Wolfgang Schreiner. *Initial Results on Modeling in PRISM Mobile Cellular Networks with Spectrum Renting*. Technical Report. Johannes Kepler University Linz, Austria: Research Institute for Symbolic Computation (RISC), Mar. 2013. URL: http://www.risc.jku.at/publications/download/risc_4705/main.pdf.

[6] Wolfgang Schreiner, Nikolaj Popov, Tamas Berczes, Janos Sztrik, and Gabor Kusper. *Applying High Performance Computing to Analyzing by Probabilistic Model Checking Mobile Cellular Networks with Spectrum Renting*. Technical Report. Johannes Kepler University Linz, Austria: Research Institute for Symbolic Computation (RISC), July 2013. URL: http://www.risc.jku.at/publications/download/risc_4705/main.pdf.

A. The PRISM Model without Spectrum Renting

```
// ------------------------------------------------------------------
// Spectrum0.prism
// A model for mobile cellular networks.
//
// The model serves as the comparison basis for the improvements
// introduced by the application of "spectrum renting" in
//
//   Tien v. Do, Patrick Wüchner, Tamas Berczes, Janos Sztrik,
//   Hermann de Meer: A New Finite-Source Queueing Model for
//   Mobile Cellular Networks Applying Spectrum Renting,
//   September 2012.
//
// Use for fastest checking the "Sparse" engine and the "Gauss-Seidel" solver
// and switch off "use compact schemes".
//
// Author: Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Copyright (C) 2013, Research Institute for Symbolic Computation
// Johannes Kepler University, Linz, Austria, http://www.risc.jku.at
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// system parameters
// ------------------------------------------------------------------

// bounds
const int K = 100;   // population size
const int n;         // number of servers/channels

// rates
const double rho;                  // normalized traffic intensity
const double mu     = 1/53.22;     // service rate
const double lambda = rho*n*mu/K; // call generation rate
const double nu     = 1;           // retrial rate
const double eta    = 1/300;       // rate of queueing users getting impatient

// probabilities
const double p_b  = 0.1;       // prob. that user gives up (-> sources)
const double p_q  = 0.5;       // prob. that user presses button (-> queue)
const double p_o  = 1-p_b-p_q; // prob. that user retries later (-> orbit)
const double p_io = 0.8;       // prob. that impatient user retries later (-> orbit)
const double p_is = 1-p_io;    // prob. that impantient user gives up (-> sources)

// ------------------------------------------------------------------
// system model
// note that the order of the modules influences model checking time
// heuristically, this seems to be the best one
// ------------------------------------------------------------------

// available servers accept requests
```

```
module Servers
  servers: [0..n] init 0;
  [sservers] servers < n -> (servers' = servers+1);
  [oservers] servers < n -> (servers' = servers+1);
  [ssources1] servers > 0 & queue = 0 ->
    servers*mu : (servers' = servers-1);
  [ssources2] servers > 0 ->
     servers*mu : true ;
endmodule

// generate requests at rate sources*lambda
module Sources
  sources: [0..K] init K;
  [sservers] sources > 0 ->
    sources*lambda : (sources' = sources-1);
  [sorbit]   sources > 0 & servers = n ->
    sources*lambda*p_o : (sources' = sources-1);
  [squeue]   sources > 0 & servers = n ->
    sources*lambda*p_q : (sources' = sources-1);
  [ssources1] sources < K -> (sources' = sources+1);
  [ssources2] sources < K -> (sources' = sources+1);
  [qsources]  sources < K -> (sources' = sources+1);
  [osources]  sources < K -> (sources' = sources+1);
endmodule

// if no server is available, requests are redirected
// with probability p_o to the orbit
formula orbit = K-(sources+servers+queue); // make variable virtual
module Orbit
  // orbit: [0..K-n] init 0;
  [sorbit]   orbit < K-n -> true;
  [qorbit]   orbit < K-n -> true;
  [oservers] orbit > 0 -> orbit*nu : true;
  [oqueue]   orbit > 0 & servers = n -> orbit*nu*p_q : true;
  [osources] orbit > 0 & servers = n -> orbit*nu*p_b : true;
endmodule

// if no server is available, requests are redirected
// with probability p_q to the queue
module Queue
  queue: [0..K-n] init 0;
  [squeue]    queue < K-n -> (queue' = queue+1);
  [oqueue]    queue < K-n -> (queue' = queue+1);
  [qorbit]    queue > 0 & servers = n ->
    queue*eta*p_io : (queue' = queue-1);
  [qsources]  queue > 0 & servers = n ->
    queue*eta*p_is : (queue' = queue-1);
  [ssources2] queue > 0 -> (queue' = queue-1);
endmodule

// ------------------------------------------------------------------
// system rewards
// ------------------------------------------------------------------
```

```
// mean number of active requests
rewards "mM"
  true : max(0, orbit)+queue+servers;
endrewards

// mean number of calls in orbit
rewards "mO"
  true : max(0, orbit);
endrewards

// mean number of calls in queue
rewards "mQ"
  true: queue;
endrewards

// mean number of active calls
rewards "mC"
  true: servers;
endrewards


// ------------------------------------------------------------------
// Spectrum0.props
// ------------------------------------------------------------------

// mean number of active requests
"mM" : R{"mM"}=? [ S ] ;

// mean number of active sources
"mK" : K-"mM" ;

// mean throughput (served and unserved)
"m1" : "mK"*lambda ;

// mean number of active calls
"mC" : R{"mC"}=? [ S ] ;

// mean goodput
"m1good" : "mC"*mu ;

// probability that arriving customer gets served
"Pgood" : "m1good"/"m1" ;

// mean response time (served and unserved)
"mT" : "mM"/"m1" ;

// mean number of idle servers
"mAS" : n-"mC" ;

// utilization of available servers
"Sutil" : "mC"/n ;

// blocking probability
"Pblock" : S=? [ servers = n ] ;
```

```
// mean queue length
"mQ" : R{"mQ"}=? [ S ] ;

// mean time spent in queue
"mTQ" : "mQ" / "m1" ;

// mean orbit length
"mO" : R{"mO"}=? [ S ] ;

// mean time spent in orbit
"mTO" : "mO" / "m1" ;
```

B. The PRISM Model with Spectrum Renting

```
// ------------------------------------------------------------------
// Spectrum.prism
// A model for mobile cellular networks applying spectrum renting.
//
// The model is described in
//
//   Tien v. Do, Patrick Wüchner, Tamas Berczes, Janos Sztrik,
//   Hermann de Meer: A New Finite-Source Queueing Model for
//   Mobile Cellular Networks Applying Spectrum Renting,
//   September 2012.
//
// Use for fastest checking the "Sparse" engine and the "Gauss-Seidel" solver
// and switch off "use compact schemes".
//
// Author: Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Copyright (C) 2013, Research Institute for Symbolic Computation
// Johannes Kepler University, Linz, Austria, http://www.risc.jku.at
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// system parameters
// ------------------------------------------------------------------

// renting tresholds
const int t1;      // block renting treshold
const int t2 = 6;  // block release treshold

// bounds
const int K = 100;   // population size
const int r = 8;     // number of servers/channels per block
const int m = 5;     // maximum number of blocks that can be rented
const int n = 2*r;   // minimum number of servers/channels
const int M = n+r*m; // maximum number of simultaneous calls
```

```
// rates
const double rho;                 // normalized traffic intensity
const double mu     = 1/53.22;    // service rate
const double lambda = rho*n*mu/K; // call generation rate
const double nu     = 1;          // retrial rate
const double eta    = 1/300;      // rate of queueing users getting impatient
const double lam_r  = 1/5;        // block renting rate
const double nu_r   = 1/7;        // block rental retrial rate
const double mu_r   = 1;          // block release reate

// probabilities
const double p_b  = 0.1;       // prob. that user gives up (-> sources)
const double p_q  = 0.5;       // prob. that user presses button (-> queue)
const double p_o  = 1-p_b-p_q; // prob. that user retries later (-> orbit)
const double p_io = 0.8;       // prob. that impatient user retries later (-> orbit)
const double p_is = 1-p_io;    // prob. that impantient user gives up (-> sources)
const double p_r  = 0.8;       // block rental success probability
const double p_f  = 1-p_r;     // block rental failure probability

// ------------------------------------------------------------------
// system model
// note that the order of the modules influences model checking time
// heuristically, this seems to be the best one
// ------------------------------------------------------------------

// number of currently available servers/channels
formula servAvail = n+blocks*r;

// blocks are rented at rate lam_r and released at rate mu_r
// renting is successful with probability p_r and fails with probability p_f
// retrying a failed attempt is performed at rate nu_r
module Blocks
  blocks: [0..m] init 0;
  trial: [0..1] init 0;
  [success1] trial = 0 & servAvail-servers <= t1 & blocks < m & queue = 0 ->
    lam_r*p_r: (blocks' = blocks+1);
  [success2] trial = 0 & servAvail-servers <= t1 & blocks < m ->
    lam_r*p_r: (blocks' = blocks+1);
  [failure] trial = 0 & servAvail-servers <= t1 & blocks < m ->
    lam_r*p_f: (trial' = 1);
  [retrial1] trial = 1 & servAvail-servers <= t1 & blocks < m & queue = 0 ->
    nu_r*p_r : (trial' = 0) & (blocks' = blocks+1) ;
  [retrial2] trial = 1 & servAvail-servers <= t1 & blocks < m ->
    nu_r*p_r : (trial' = 0) & (blocks' = blocks+1) ;
  [interrupt] trial = 1 & servAvail-servers > t1  ->
    9999 : (trial' = 0); // "immediately"
  [release] servAvail-servers >= t2+r & blocks > 0 ->
    mu_r : (blocks' = blocks-1);
endmodule

// available servers accept requests
module Servers
  servers: [0..M] init 0;
  [sservers] servers < servAvail -> (servers' = servers+1);
```

```
  [oservers] servers < servAvail -> (servers' = servers+1);
  [success2] servers < M         -> (servers' = servers+1);
  [retrial2] servers < M         -> (servers' = servers+1);
  [ssources1] servers > 0 & queue = 0 ->
    servers*mu : (servers' = servers-1);
  [ssources2] servers > 0 ->
     servers*mu : true ;
endmodule

// generate requests at rate sources*lambda
module Sources
  sources: [0..K] init K;
  [sservers] sources > 0 ->
    sources*lambda : (sources' = sources-1);
  [sorbit]   sources > 0 & servers = servAvail ->
    sources*lambda*p_o : (sources' = sources-1);
  [squeue]   sources > 0 & servers = servAvail ->
    sources*lambda*p_q : (sources' = sources-1);
  [ssources1] sources < K -> (sources' = sources+1);
  [ssources2] sources < K -> (sources' = sources+1);
  [qsources]  sources < K -> (sources' = sources+1);
  [osources]  sources < K -> (sources' = sources+1);
endmodule

// if no server is available, requests are redirected
// with probability p_o to the orbit
formula orbit = K-(sources+servers+queue); // make variable virtual
module Orbit
  // orbit: [0..K-n] init 0;
  [sorbit]   orbit < K-n -> true;
  [qorbit]   orbit < K-n -> true;
  [oservers] orbit > 0 -> orbit*nu : true;
  [oqueue]   orbit > 0 & servers = servAvail -> orbit*nu*p_q : true;
  [osources] orbit > 0 & servers = servAvail -> orbit*nu*p_b : true;
endmodule

// if no server is available, requests are redirected
// with probability p_q to the queue
module Queue
  queue: [0..K-n] init 0;
  [squeue]    queue < K-n -> (queue' = queue+1);
  [oqueue]    queue < K-n -> (queue' = queue+1);
  [qorbit]    queue > 0 & servers = servAvail ->
    queue*eta*p_io : (queue' = queue-1);
  [qsources]  queue > 0 & servers = servAvail ->
    queue*eta*p_is : (queue' = queue-1);
  [ssources2] queue > 0 -> (queue' = queue-1);
  [success2]  queue > 0 -> (queue' = queue-1);
  [retrial2]  queue > 0 -> (queue' = queue-1);
endmodule

// ------------------------------------------------------------------
// system rewards
// ------------------------------------------------------------------
```

```
// mean number of active requests
rewards "mM"
  true : max(0, orbit)+queue+servers;
endrewards

// mean number of calls in orbit
rewards "mO"
  true : max(0, orbit);
endrewards

// mean number of calls in queue
rewards "mQ"
  true: queue;
endrewards

// mean number of active calls
rewards "mC"
  true: servers;
endrewards

// mean number of active blocks
rewards "mB"
  true: blocks;
endrewards


// ------------------------------------------------------------------
// Spectrum.props
// ------------------------------------------------------------------

// mean number of active requests
"mM" : R{"mM"}=? [ S ] ;

// mean number of active sources
"mK" : K-"mM" ;

// mean throughput (served and unserved)
"m1" : "mK"*lambda ;

// mean number of active calls
"mC" : R{"mC"}=? [ S ] ;

// mean goodput
"m1good" : "mC"*mu ;

// probability that arriving customer gets served
"Pgood" : "m1good"/"m1" ;

// mean response time (served and unserved)
"mT" : "mM"/"m1" ;

// mean number of rented blocks
"mB" : R{"mB"}=? [ S ] ;
```

```
// mean number of available servers
"mS" : n+"mB"*r ;

// mean number of idle servers
"mAS" : "mS"-"mC" ;

// utilization of available servers
"Sutil" : "mC"/"mS" ;

// blocking probability
"Pblock" : S=? [ servers = servAvail ] ;

const int B;

// probability that B blocks are partially utilized
"Pb" : S=? [ n+r*(B-1) < servers & servers <= n+r*B ] ;

// mean queue length
"mQ" : R{"mQ"}=? [ S ] ;

// mean time spent in queue
"mTQ" : "mQ" / "m1" ;

// mean orbit length
"mO" : R{"mO"}=? [ S ] ;

// mean time spent in orbit
"mTO" : "mO" / "m1" ;

const int d;

// average profit rate
"APR" : "mC" - (r/d) * "mB" ;
```

C. The Parallel Execution Script

```
#!/bin/sh

# the program locations
export PRISM_JAVA="prism-java"
PRISM="prism"
PARALLEL="./parallel"
TIME="time"

# the input/output locations
MODELFILE="Spectrum.prism"
MODELFILE0="Spectrum0.prism"
MODELFILE2="Spectrum2.prism"
MODELFILE3="Spectrum3.prism"
PROPSFILE="Spectrum.props"
PROPSFILE0="Spectrum0.props"
RESULTDIR="Results"
```

```
LOGDIR="Logfiles"
LOGFILE="LOGFILE"

# the checker settings
PRISMOPTIONS="-sparse -gaussseidel -nocompact"

# the number of processes to be used
for PROC in 1 2 4 8 16 32 ; do

(

# the properties to be checked and the parameters for the experiment

# Figure 2
for PROPERTY in Pblock mO mTO mQ mTQ mAS ; do
  for RHO in $(seq 0.6 0.5 4.6) ; do
    for N in 8 16 24 32 ; do
      echo "$PRISM $PRISMOPTIONS $MODELFILE0 $PROPSFILE0 -prop $PROPERTY \
        -const rho=$RHO,n=$N \
        -exportresults $RESULTDIR/Fig2-$PROPERTY-$N-$RHO \
        > $LOGDIR/Fig2-$PROPERTY-$N-$RHO"
    done
  done
done

# Figure 3
for PROPERTY in Pblock mO mTO mB mQ mTQ mC mAS ; do
  for RHO in $(seq 0.6 0.5 4.6) ; do
    for T1 in $(seq 1 1 4) ; do
      echo "$PRISM $PRISMOPTIONS $MODELFILE $PROPSFILE -prop $PROPERTY \
        -const rho=$RHO,t1=$T1 \
        -exportresults $RESULTDIR/Fig3-$PROPERTY-$T1-$RHO \
        > $LOGDIR/Fig3-$PROPERTY-$T1-$RHO"
    done
  done
done

# Figure 4
PROPERTY="Pb"
for B in $(seq 1 1 4) ; do
  for RHO in $(seq 0.6 0.5 4.6) ; do
    for T1 in $(seq 1 1 4) ; do
      echo "$PRISM $PRISMOPTIONS $MODELFILE $PROPSFILE -prop $PROPERTY \
        -const B=$B,rho=$RHO,t1=$T1 \
        -exportresults $RESULTDIR/Fig4-$PROPERTY-$B-$T1-$RHO \
        > $LOGDIR/Fig4-$PROPERTY-$B-$T1-$RHO"
    done
  done
done

# Figure 5
RHO="0.6"
for PROPERTY in mB Pblock mQ mO mTQ mTO ; do
  for T1 in $(seq 0 1 4) ; do
```

```
    for T2 in $(seq 5 1 8) ; do
      echo "$PRISM $PRISMOPTIONS $MODELFILE2 $PROPSFILE -prop $PROPERTY \
        -const rho=$RHO,t1=$T1,t2=$T2 \
        -exportresults $RESULTDIR/Fig5-$PROPERTY-$T1-$T2 \
        > $LOGDIR/Fig5-$PROPERTY-$T1-$T2"
    done
  done
done

# Figures 6-7
PROPERTY="APR"
for RHO in 0.6 4.6 ; do
  for T1 in $(seq 0 1 4) ; do
    for T2 in 5 8 ; do
      for D in 1 2 4 8 ; do
        echo "$PRISM $PRISMOPTIONS $MODELFILE2 $PROPSFILE -prop $PROPERTY \
          -const rho=$RHO,t1=$T1,t2=$T2,d=$D \
          -exportresults $RESULTDIR/Fig67-$PROPERTY-$T1-$T2-$D-$RHO \
          > $LOGDIR/Fig67-$PROPERTY-$T1-$T2-$D-$RHO"
      done
    done
  done
done

# Figure 8, T1 apparently 2
PROPERTY="APR"
T1=2
for RHO in $(seq 0.6 0.5 4.6) ; do
  for T2 in 5 8 ; do
    for D in 1 8 ; do
      echo "$PRISM $PRISMOPTIONS $MODELFILE2 $PROPSFILE -prop $PROPERTY \
        -const rho=$RHO,t1=$T1,t2=$T2,d=$D \
        -exportresults $RESULTDIR/Fig8-$PROPERTY-$T2-$D-$RHO \
        > $LOGDIR/Fig8-$PROPERTY-$T2-$D-$RHO"
      done
  done
done

# Figures 9,10
PROPERTY="APR"
T1=2
T2=5
D=2
for PROPERTY in "APR" "mC" ; do
  PO=0.2
  PIO=0.4
  for RHO in $(seq 4.55 0.01 4.6) ; do
     echo "$PRISM $PRISMOPTIONS $MODELFILE3 $PROPSFILE -prop $PROPERTY \
       -const rho=$RHO,t1=$T1,t2=$T2,d=$D,p_o=$PO,p_io=$PIO \
       -exportresults $RESULTDIR/Fig910-$PROPERTY-$PO-$PIO-$RHO \
       > $LOGDIR/Fig910-$PROPERTY-$PO-$PIO-$RHO"
     done
  PO=0.4
  PIO=0.8
```

```
  for RHO in $(seq 4.55 0.01 4.6) ; do
     echo "$PRISM $PRISMOPTIONS $MODELFILE3 $PROPSFILE -prop $PROPERTY \
       -const rho=$RHO,t1=$T1,t2=$T2,d=$D,p_o=$PO,p_io=$PIO \
       -exportresults $RESULTDIR/Fig910-$PROPERTY-$PO-$PIO-$RHO \
       > $LOGDIR/Fig910-$PROPERTY-$PO-$PIO-$RHO"
  done
  PO=0.000000001
  PIO=0.000000001
  for RHO in $(seq 4.55 0.01 4.6) ; do
     echo "$PRISM $PRISMOPTIONS $MODELFILE3 $PROPSFILE -prop $PROPERTY \
       -const rho=$RHO,t1=$T1,t2=$T2,d=$D,p_o=$PO,p_io=$PIO \
       -exportresults $RESULTDIR/Fig910-$PROPERTY-$PO-$PIO-$RHO \
       > $LOGDIR/Fig910-$PROPERTY-$PO-$PIO-$RHO"
  done
done

# execute the experiments in parallel with PROC processes
) | $TIME -p $PARALLEL $PROC > $LOGDIR/$LOGFILE 2>&1

done
```