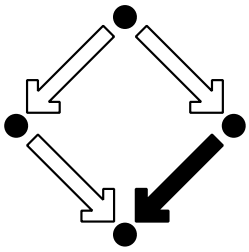
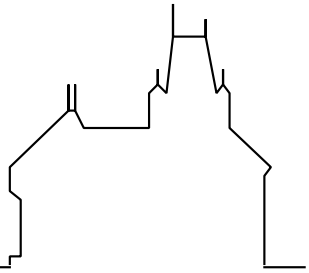


RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe



On the Probabilistic Model Checking of Cognitive Radio Networks and Cognitive Information Systems

Wolfgang SCHREINER Tamas BERCZES
Janos SZTRIK, Hamza NEMOUCHI

(February 2018)

RISC-Linz Report Series No. 18-04

Supported by: Stiftung Aktion Österreich-Ungarn project 96öu8
"Engineering Information Technology Modelling and Design of Cognitive Radio Networks"

On the Probabilistic Model Checking of Cognitive Radio Networks and Cognitive Infocommunication Systems*

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

Tamás Bérczes and János Sztrik and Hamza Nemouchi
Department of Informatics Systems and Networks, Faculty of Informatics
University of Debrecen, Debrecen, Hungary
berczes.tamas@inf.unideb.hu, sztrik.janos@inf.unideb.hu
nemouchih@gmail.com,

February 27, 2018

Abstract

We report on the usage of the probabilistic model checker PRISM to validate respectively falsify previously published results on the performance of cognitive radio networks and other cognitive infocommunication systems. For this purpose, we construct formal system models in the PRISM modeling language as Continuous Time Markov Chains (CTMCs) that are analyzed with the help of queries formulated in a variant of continuous stochastic logic (CSL). It is shown that many results can be accurately reproduced but also that a few previously reported results are clearly erroneous. Furthermore, we report several deviations from previously reported results where the reasons are unclear and need further investigation. Here a major problem is that the systems that have been analyzed in literature are usually just described in informal language which leaves ample room for interpretation. This makes the reconstruction of corresponding formal system models and the reproduction of performance measures a difficult task and limits the long-term scientific value of the reported results.

*Supported by the Stiftung Aktion Österreich-Ungarn project 96öu8 "Engineering Information Technology Modelling and Design of Cognitive Radio Networks".

Contents

1 Introduction 3

2 A Cognitive Radio Network 3

3 A Cognitive Radio Network with Collisions 5

4 A Cognitive Radio Network with Non-reliable Servers and Collisions 6

5 A Cognitive Communication System with Network Service Breakdown 7

6 A Multilayered Cognitive Communication System with Network Service Breakdown 7

7 Conclusions 9

A A Cognitive Radio Network 11

B A Cognitive Radio Network with Collisions 13

C A Cognitive Radio Network with Non-reliable Servers and Collisions 16

D A Cognitive Communication System with Network Service Breakdown 19

E A Multilayered Cognitive Communication System with Network Series Breakdown 21

1 Introduction

Cognitive radio networks represent a special kind of wireless communication systems that aim to enhance the utilization of the radio frequency spectrum by dividing the customers in two groups: the "primary customers" have licensed some part of the spectrum to which they get priority access; the "secondary customers" use another part of the spectrum but may opportunistically get also access to the primary spectrum. However, the primary customers must not experience interference from the secondary ones; thus the secondary customers must use "cognitive" devises that sense the primary spectrum such that on the one hand this spectrum is only used if there are no primary customers and on the other hand the primary spectrum is released as soon as primary customers appear.

In this paper, we re-investigate some of the authors' prior research on cognitive systems in general and cognitive radio networks in particular. This research has yielded analytical performance models that were numerically validated, either by simulation (with the help of manually crafted C++ programs) or by use of the performance evaluation tool MOSEL-2. However, since these results were never independently validated, there always exists the danger of errors respectively inaccuracies in the derived results respectively in the presentation of these results; this is the more the case since the corresponding programs respectively tool-based models were never publicly released.

We base our validation on PRISM [7, 4], a probabilistic model checker that can be used for the formal modeling and analysis of systems that exhibit random, probabilistic, or stochastic behavior. In particular, this tool allows to model systems in the simple state-based PRISM language as Continuous Time Markov Chains (CTMC) and to query the properties/measures of such a model in a variant of Continuous Stochastic Logic (CSL). Already in the past we have variously used PRISM for this purpose (e.g. in [8]) and often been able to find discrepancies and error in previously published results. This paper is an attempt to repeat to apply our experience to the domain of cognitive radio networks.

As the basis of our investigations, we take the recent papers [1, 9, 5, 6] on various models of cognitive radio networks and the earlier publications [2, 3] on cognitive information systems of a somewhat different kind. Each of the following chapters 2, 3, 4, 5, and 6 presents a PRISM model corresponding to a model described in one of these publications, and analyzes the correspondence respectively differences (with the corresponding formal models listed in Appendices A, B, C, D, and E). Our investigations are still in an early state; preliminary conclusions are given in Section 7.

2 A Cognitive Radio Network

Appendix A gives the PRISM model of a cognitive radio network discussed in [1, 9]. This system consists of two interdependent subsystems:

- The *primary subsystem* contains a source of N_1 primary users (PUs) that request at rate λ_1 access to a primary channel service (PCS) that processes requests at rate μ_1. If the PCS is busy with another PU, the user joins a queue of PUs waiting for service; however, if the PCS is occupied by a secondary user (SU, see below), this SU is evicted to the secondary subsystem and the PU takes over the PCS.
- The *secondary subsystem* contains a source of N_2 secondary users (SUs) that request at rate λ_2 access to a secondary channel service (SCS) that processes requests at rate μ_2. If the SCS is occupied by another SU, the status of the PCS is investigated; if the PCS happens to be free, the SU is directed to the PCS. If also the PCS is busy, then the SU joins the "orbit", an area of waiting SUs each of which re-attempts access to the SCS/PCS at rate ν.

The execution of a service in the PCS/SCS (the transmission of a radio package) may fail with a probability p; in that case the corresponding user has to repeat its request to service access.

The PRISM model implements this system by modules `Primary` (the source of PUs), `Secondary` (the source of SUs), `PCS`, `SCS`, and `Orbit`. In a slight deviation from above description, the model is a bit simplified in the

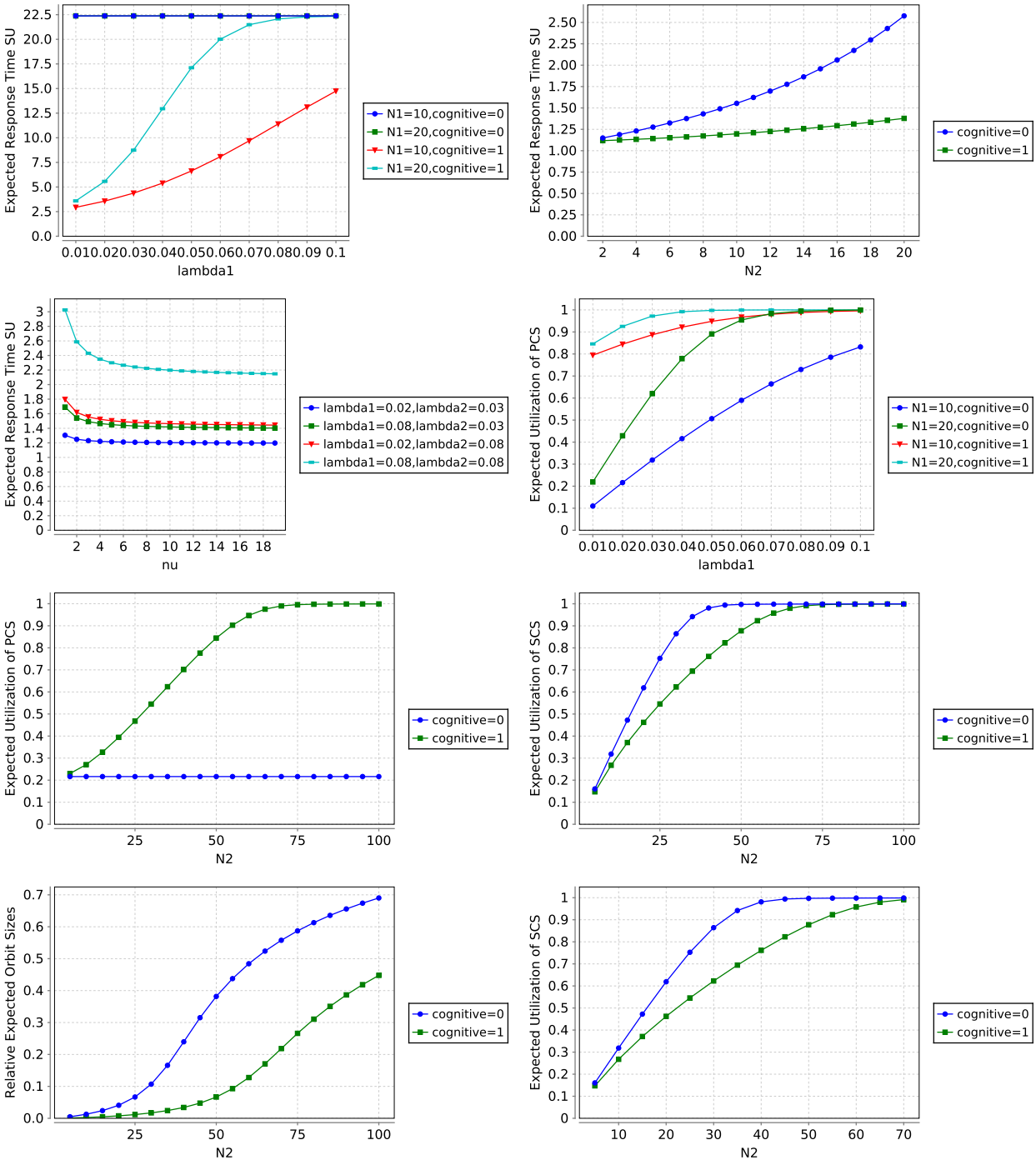


Figure 1: A Cognitive Radio Network (see Fig. 2–9 from [1])

following way: if the execution of a service fails, the user does not leave the occupying service but immediately repeats the execution. From the point of the model, this is identical to letting the job join the queue/orbit and taking another job from that queue/orbit for execution; the only difference to above description is that, if the execution of a SU in the PCS fails, the SU immediately continues execution in the PCS rather than checking whether the SCS is empty and to continue in the SCS, if this is the case.

Figure 1 demonstrates the results of the analysis of this model with PRISM using the same values for the model parameters that were used in [1] in order to produce corresponding results with the performance evaluation tool MOSEL-2. In most cases, we are very well able to reproduce with PRISM the results reported there for MOSEL-2 in Figures 2–9, with the following exceptions:

- We are not able to reproduce the results presented in Figure 3 (expected response time SU), the values are one to two orders of magnitude off.
- Figures 6–9 present for small values of N_2 ($N_2 = 5$) significantly different results. Also while the overall shapes of the figures seem to match the reported results, they are actually slightly off.

We believe that the difference in Figure 3 is likely an error in [1] in that the figure there was produced with different parameter values. However, the deviations in Figures 6–9 seem to indicate subtle differences between the PRISM model and the MOSEL model; these need further investigation.

3 A Cognitive Radio Network with Collisions

Appendix B gives the PRISM model for the system described in [5], which extends the cognitive ratio network by the concept of "collisions":

- If a SU unit is evicted from the PCS (due to the arrival of a PU), it immediately attempts to use the SCS; if the SCS however, was already in use by another SU, this causes a collision where both SUs move from the SCS to the orbit.
- Likewise, if a newly arriving SU finds the PCS busy, it attempts to use the SCS, which may cause a collision with another SU such that both move to the orbit (however, since the use of the PCS is only attempted, if the SCS is empty, it remains unclear how realistic the modeling of this case actually is).

In [5], numerical results are produced by stochastic simulation (using a manually crafted C++ program), not only for exponentially distributed service times, but also for hypo-exponential and hyper-exponential distributions. In the following use of PRISM, we however, consider only exponential distributions.

Figure 2 immediately demonstrates that our results significantly deviate from those reported in [5]. The curve in the left diagram should match the exponential case of Figure 2 in that paper but actually gives for small values of λ_1 significantly higher values, already approaching with $\lambda_1 \geq 0.08$ the expected response time 40, which is in [5] only reached with $\lambda_1 \geq 0.2$.

Likewise, the curve in the right diagram is a flat line with a value of about 10, while the corresponding Figure 5 in [5] predicts a value of about 3.3. Actually, since in this diagram the second subsystem does not play a role, we can calculate the expected value, the mean response time of a PU, by manual calculation: it is the value T denoted by the following equations:

$$
\begin{aligned}
T &= \overline{N_1}/((N_1 - \overline{N_1}) \cdot \lambda_1) \\
\overline{N_1} &= N_1 - U_s/\rho \\
\rho &= \lambda_1/\mu_1 \\
U_s &= 1 - P_0 \\
P_0 &= \frac{1}{N_1!} \Big/ \sum_{i=0}^{N_1} \frac{\rho^i}{(N_1 - i)!}
\end{aligned}
$$

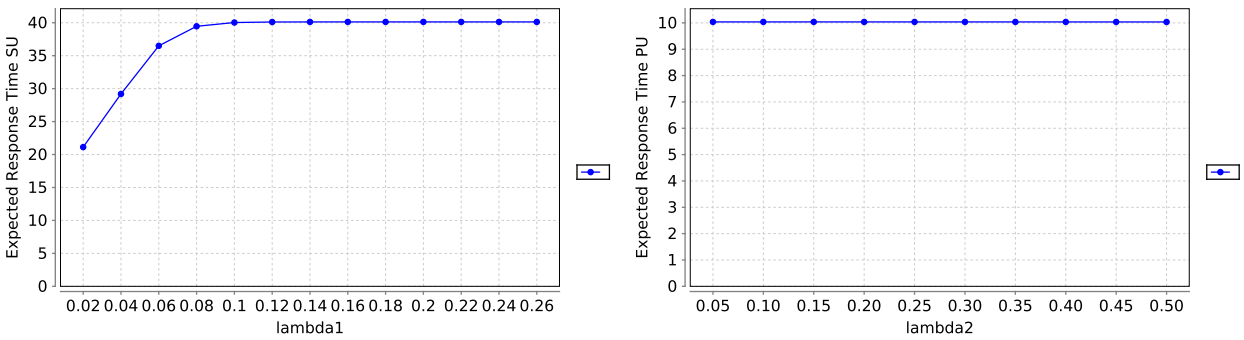


Figure 2: A Cognitive Radio Network with Collisions (see Fig. 2 and 5 from [5])

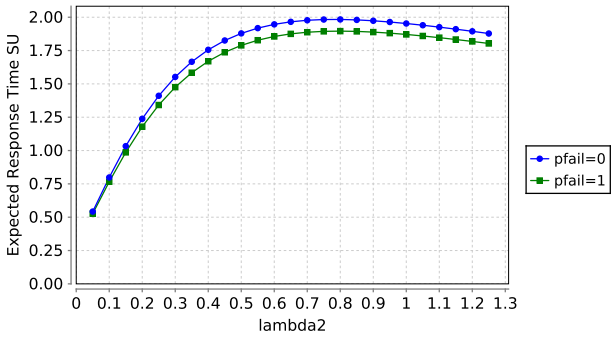


Figure 3: A Cognitive Radio Network with Non-reliable Servers and Collisions (see Fig. 2 from [6])

A calculation with the mathematical software system Mathematica demonstrates that the expected value is indeed close to 10 (the numerical accuracy of the value computed by PRISM is accurate with several digits).

Since therefore the numerical results in [5] are questionable, we omit further comparisons until a more detailed investigation and clarification of the deviations.

4 A Cognitive Radio Network with Non-reliable Servers and Collisions

Appendix C gives the PRISM model for the system described in [6], which extends the cognitive ratio network with collisions by the concept of "non-reliable servers":

- The PCS may fail with an exponentially distributed rate γ_1; if the PCS fails while executing a PU, the PU goes back to the service queue; if the PCS fails while executing a SU, the SU attempts access to the SCS (which may cause a collision with another currently executing SCS).
- The SCS may fail with an exponentially distributed rate γ_2; if the SCS fails while executing a SU, the interrupted SU may start executing at the PCS, provided that this PCS is free; otherwise the SU goes to the orbit.

A failed PCS is repaired with rate σ_1, a failed SCS is repaired with rate σ_2.

Also in [6], numerical results are produced by stochastic simulation (using a manually crafted C++ program). Unfortunately, our Figure 3 demonstrates that our results significantly deviate from the results reported there in Figure 2. While the curves generally have the same shape and values of similar order of magnitude, the

actual values are significantly different: for $\lambda_2 = 0.1$, we get a value of about 0.5 while [6] reports a value of a bit less than 0.7; our curves reach a maximum of abut 2.0 for $\lambda_2 \simeq 0.75$, while [6] reports a maximum of about 1.9 for $\lambda_2 \simeq 2.4$.

We are thus not able to reproduce the results of [6]; the discrepancies need further investigation.

5 A Cognitive Communication System with Network Service Breakdown

Appendix D lists the PRISM model for a cognitive infocommunication system with network service breakdown reported in [2]; this model consists of one server with two request queues of total maximum capacity B and two classes of users:

- A class of N_2 "normal" users that generate service requests at rate λ_2. If a normal user finds the service busy, it attempts to enter the "normal" queue. If, however, the maximum capacity B is reached, the service is rejected.
- A class of N_1 "intelligent" users that generate service requests at rate λ_1. If an intelligent user finds the service busy, it enters the "intelligent" queue. If the maximum capacity B is reached, the user enters the orbit and retries the service request at rate ν.

A server executes service requests from both queues with equal probability. The quality of the server, however, degrades with rate δ. If the server is still fully operational, it may with probability p degrade to a limited state, and with probability $1 - p$ totally fail; if the server is already in a limited state, it may (again with rate δ) fail. If the server is fully operational, it processes requests with rate μ_1; in limited state, however, requests are processed with rate $\mu_2 < \mu_1$. If the server has totally failed, it does not process any requests (the remainder of the system, however, may progress as usual). A server in limited state is repaired with rate β_1; a failed server is repaired with rate β_2 (in the following, however, a single repair rate β is assumed).

Figure 4 present the results derived with PRISM mimicking the Figures 3–7 from [2]. Most results very accurately confirm the previously reported ones; only the expected response time are in our model slightly lower than the originally reported ones, as well as the wasted times; this seems to hint to a slight difference in the models that requires further investigation.

6 A Multilayered Cognitive Communication System with Network Service Breakdown

Appendix E lists the PRISM model for a multilayered cognitive infocommunication system with network service breakdown reported in [3]; this model modifies the model previously investigated in [3] in the way in which the service operates:

- The server runs a pipeline of three service stages that operate with rates μ_1, μ_2, and μ_3; only one stage is active at every time, i.e., a new request may enter stage 1 only after the processing of the previous request has been completed by stage 3.
- Each stage may fail with rate δ; a failed stage is repaired with rate β. If a stage fails, it does not accept any further request or continue to process the current one; otherwise, however, the failure of a stage does not influence the operation of the rest of the system.

Figure 5 present the results derived with PRISM mimicking the Figures 2–5 from [3]. Here we are not able to reproduce the previously reported results; while all curves have similar shapes, the absolute values are considerably different.

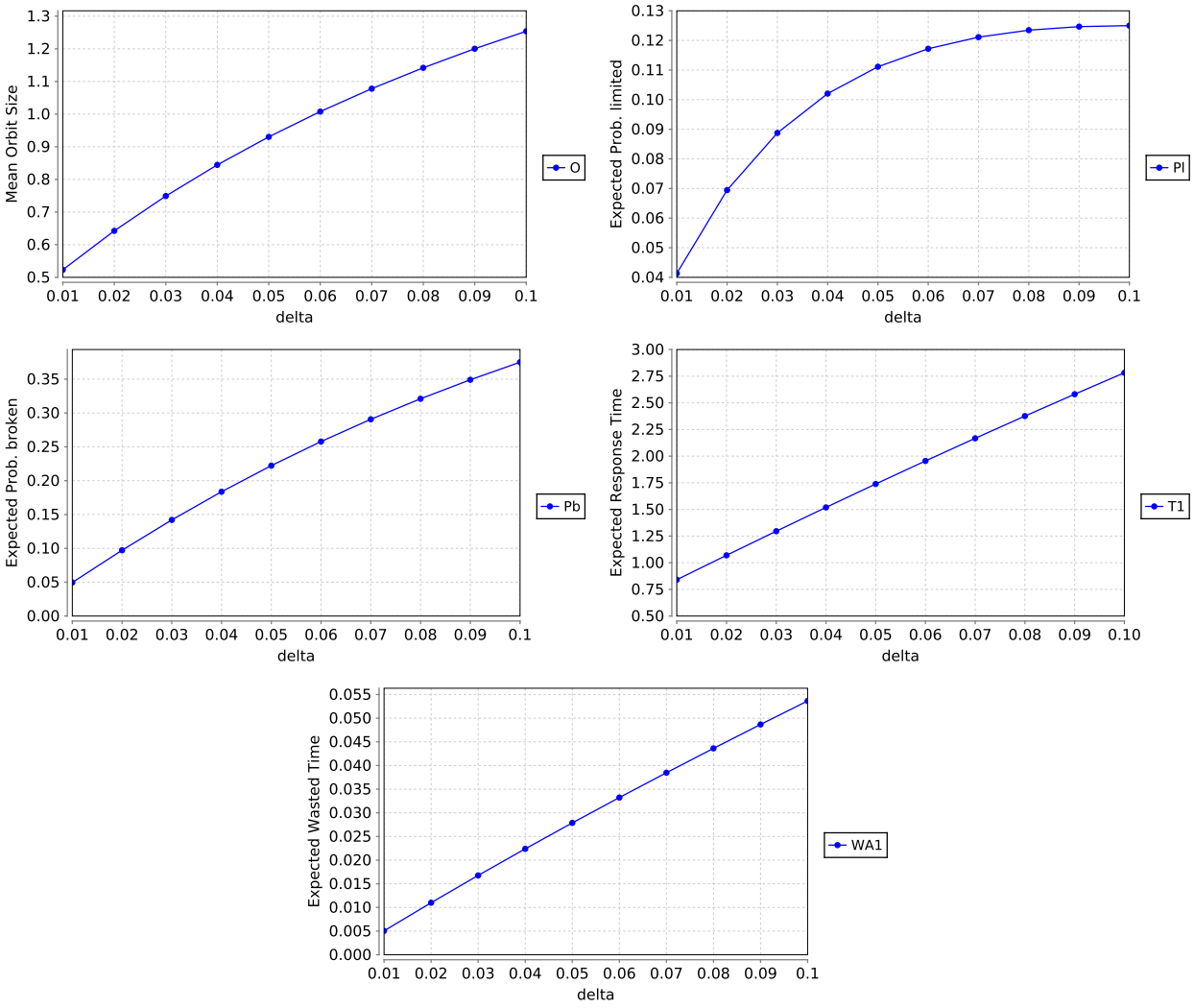


Figure 4: A Cognitive Communication System with Network Service Breakdown (see Fig. 3–7 from [2])

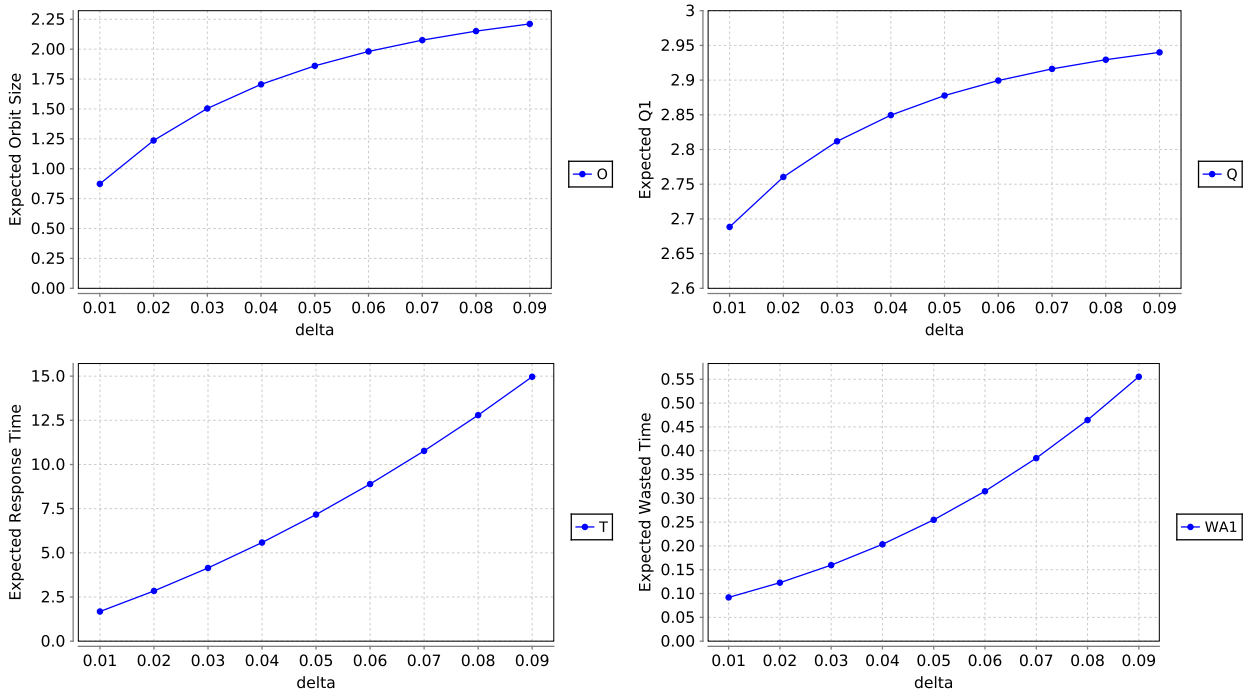


Figure 5: A Multilayered Cognitive Communication System with Network Service Breakdown (see Fig. 2–5 from [3])

As for Figure 2, the results reported in [3] cannot be correct, because they report a mean orbit size of more than 5.5 that is larger than the number $N_1 = 3$ of intelligent users; apparently here different parameters than the reported ones have been used.

As for the other differences, however, we suspect the failure to be in our understanding of the model. The description of how to deal with failure situations given in [3] leaves ample room for interpretation in various situations; most likely, our understanding of the system is thus different from the originally intended one.

7 Conclusions

The attempts to validate the previously published results on the performance of various variants of cognitive radio networks and cognitive infocommunication systems has produced mixed results:

- We were mostly able to successfully reproduce with PRISM the results about the radio network reported in [1] and the infocommunication system reported in [2] that were produced in those papers with the help of MOSEL-2. Apart from one case, the deviations seem to be mostly due to subtle differences in the models.
- We were not able to reproduce the results about the infocommunication system reported in [3]. Apart from one case of a clear error in that paper, the differences are most likely due to a misunderstanding of the complex but ambiguous system description which leaves ample room for interpretation such that the PRISM model most likely is different from the original intention.
- We were not able to reproduce the results about the radio networks reported in [5] and [6] that were produced in those papers with the help of manually crafted simulation programs. At least in one case this is due to a clear error in that paper. As for the other results, the reasons remain unclear and need further investigation.

As these preliminary investigations show, it is often not easily possible to reproduce previously published results on the performance of computing systems. The main problem is that the models are typically described by informal prose which may have unclear and ambiguous interpretations; mostly there are no references (e.g. download links) to the actual formal models used to produce numerical results (analytical equations in the language of a mathematical software systems, or system models in tools like MOSEL-2 or PRISM). The present report therefore explicitly lists all PRISM models used for independent usage and checking. The authors will be glad to hear about any errors and/or corrections.

References

[1] Bela Almasi, Tamas Berczes, Attila Kuki, Janos Sztrik, and Jinting Wang. "Performance Modeling of Finite-Source Cognitive Radio Networks". In: *Acta Cybernetica* 22 (2016), pp. 617–631. DOI: 10.14232/actacyb.22.3.2016.5.

[2] Attila Kuki, Tamás Bérczes, Bela Almási, and János Sztrik. "A queueing model to study the effect of network service breakdown in a CogInfoCom system". In: *4th IEEE International Conference on Cognitive Infocommunications (CogInfoCom 2013)*. Budapest, Hungary, December 2–5, 2013, pp. 205–210. DOI: 10.1109/CogInfoCom.2013.6719242.

[3] Attila Kuki, Tamás Bérczes, Bela Almási, and János Sztrik. "Investigating the effect of network service breakdown in multilayered cognitive communication system". In: *5th IEEE International Conference on Cognitive Infocommunications (CogInfoCom 2014)*. Vietri sul Mare, Italy, November 5–7, 2014, pp. 237–241. DOI: 10.1109/CogInfoCom.2014.7020453.

[4] 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. LNCS. Springer, 2011, pp. 585–591.

[5] Hamza Nemouchi and János Sztrik. "Performance Evaluation of Finite-Source Cognitive Radio Networks with Collision Using Simulation". In: *8th IEEE International Conference on Cognitive Infocommunications (CogInfoCom 2017)*. Debrecen, Hungary, September 11–14, 2017, pp. 127–131. DOI: 10.1109/CogInfoCom.2017.8268228.

[6] Hamza Nemouchi and János Sztrik. "Performance Simulation of Non-reliable Servers in Finite-Source Cognitive Radio Networks with Collision". In: *Information Technologies and Mathematical Modelling - Queueing Theory and Applications (ITMM 2017)*. Vol. 800. Communications in Computer and Information Science. Kazan, Russia, September 29 – October 3: Springer, 2017, pp. 194–203. DOI: 10.1007/978-3-319-68069-9_16.

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

[8] Wolfgang Schreiner, Tamas Berczes, and Janos Sztrik. "Probabilistic Model Checking on HPC Systems for the Performance Analysis of Mobile Networks". In: *Annales Mathematicae et Informaticae* 43 (2014), pp. 123–144.

[9] János Sztrik, Tamás Bérczes, Hamza Nemouchi, and Agassi Melikov. "Performance Modeling of Finite-Source Cognitive Radio Networks Using Simulation". In: *Distributed Computer and Communication Networks (DCCN 2016)*. Vol. 678. Communications in Computer and Information Science. Moscow, Russia, November 21–25: Springer, 2016, pp. 64–73. DOI: 10.1007/978-3-319-51917-3_7.

A A Cognitive Radio Network

```
// ------------------------------------------------------------------
// CognitiveRadio1.prism
// Finite-source cognitive radio networks.
//
// This implements the model described in
//
//   Bela Ahmasi, Tamas Berczes, Attila Kuki, Janos Sztrik, Jinting Wang:
//   "Performance Modeling of Finite-Source Cognitive Radio Networks",
//   Acta Cybernetica 22 (2016), 617-631.
//
// (c) 2018, Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// parameters
// ------------------------------------------------------------------

const int N1; // = 20; // number of primary users (PUs)
const int N2; // = 50; // number of secondary users (SUs)

const double lambda1; // = 0.1; // generation rate of PUs
const double lambda2; // = 0.1; // generation rate of SUs

const double mu1; // = 1; // service intensity at primary channel service (PCS)
const double mu2; // = 1; // service intensity at secondary channel service (SCS)

const double nu; // = 20; // retrial intensity

const double p;  // = 0.1; // probability of service failure

const int cognitive; // = 1, if cognitive mode is switched on

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module Primary
  k1: [0..N1] init N1;
  [pripcs] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [priscs] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [priorb] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [pcspri] k1 < N1 ->              (k1' = k1+1);
endmodule

module Secondary
  k2: [0..N2] init N2;
  [secscs] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [secpcs] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [secorb] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [scssec] k2 < N2 ->              (k2' = k2+1);
  [pcssec] k2 < N2 ->              (k2' = k2+1);
endmodule

module PCS
```

```
  q: [0..N1-1] init 0;
  y: [0..2] init 0; // 0: idle, 1: busy with PU, 2: busy with SU
  [pripcs] y = 0                          ->             (y' = 1);
  [pripcs] q < N1-1 & y = 1               ->             (q' = q+1);
  [priscs] y = 2                          ->             (y' = 1);
  [priorb] y = 2                          ->             (y' = 1);
  [secpcs] c = 1 & y = 0 & cognitive = 1 ->              (y' = 2);
  [orbpcs] c = 1 & y = 0 & cognitive = 1 ->              (y' = 2);
  [pcspri] y = 1 & q = 0                  -> mu1*(1-p) : (y' = 0);
  [pcspri] y = 1 & q > 0                  -> mu1*(1-p) : (q' = q-1);
  [pcssec] y = 2                          -> mu1*(1-p) : (y' = 0);
endmodule

module SCS
  c: [0..1] init 0; // 0: idle, 1: busy with SU
  [secscs] c = 0 ->             (c' = 1);
  [priscs] c = 0 ->             (c' = 1);
  [orbscs] c = 0 ->             (c' = 1);
  [scssec] c = 1 -> mu2*(1-p) : (c' = 0);
endmodule

module Orbit
  o: [0..N2-1] init 0;
  [orbscs] o > 0                                      -> o*nu : (o' = o-1);
  [orbpcs] o > 0                                      -> o*nu : (o' = o-1);
  [secorb] o < N2-1 & c = 1 & (y != 0 | cognitive = 0) ->       (o' = o+1);
  [priorb] o < N2-1 & c = 1                           ->        (o' = o+1);
endmodule

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

// average number of active primary users (= N1-M1)
rewards "K1"
  true : k1;
endrewards

// average number of jobs of primary users in network (= N1-K1)
rewards "M1"
  true : q + ( y = 1 ? 1 : 0 );
endrewards

// average number of active secondary users (= N2-M2)
rewards "K2"
  true : k2;
endrewards

// average number of jobs of secondary users in network (= N2-K2)
rewards "M2"
  true : o + c + ( y = 2 ? 1 : 0 );
endrewards

// utilization of PCS
rewards "U1"
  y != 0 : 1;
endrewards

// utilization of SCS
rewards "U2"
  c != 0 : 1;
```

```
endrewards

// orbit size
rewards "O"
  true : o;
endrewards

// ------------------------------------------------------------------
// end of model
// ------------------------------------------------------------------

// ------------------------------------------------------------------
// CognitiveRadio1.csl
// ------------------------------------------------------------------

"K1": R{"K1"}=? [ S ]
"T1": (N1-"K1")/(lambda1*"K1");

"K2": R{"K2"}=? [ S ]
"T2": (N2-"K2")/(lambda2*"K2");

"M1": R{"M1"}=? [ S ]
"K1M1": "K1"+"M1";

"M2": R{"M2"}=? [ S ]
"K2M2": "K2"+"M2";

"U1": R{"U1"}=? [ S ]
"U2": R{"U2"}=? [ S ]

"O": R{"O"}=? [ S ]
"N2O": "O"/N2
```

B A Cognitive Radio Network with Collisions

```
// ------------------------------------------------------------------
// CognitiveRadio2.prism
// Finite-source cognitive radio networks with collisions.
//
// This implements the model described in
//
//   Hamza Nemouchi, Janos Sztrik:
//   "Performance Evaluation of Finite-Source Cognitive Radio
//   Networks", 8th IEEE International Conference on Cognitive
//   Infocommunications (CogInfoCom 2017), September 11-14, 2017,
//   Debrecen, Hungary.
//
// (c) 2018, Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// parameters
// ------------------------------------------------------------------

const int N1; // = 20; // number of primary users (PUs)
```

```
const int N2; // = 50; // number of secondary users (SUs)

const double lambda1; // = 0.1; // generation rate of PUs
const double lambda2; // = 0.1; // generation rate of SUs

const double mu1; // = 1; // service intensity at primary channel service (PCS)
const double mu2; // = 1; // service intensity at secondary channel service (SCS)

const double nu; // = 20; // retrial intensity

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module Primary
  k1: [0..N1] init N1;
  [pripcs] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [priscs] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [priorb] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [pcspri] k1 < N1 ->              (k1' = k1+1);
endmodule

module Secondary
  k2: [0..N2] init N2;
  [secscs] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [secpcs] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [secorb] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [scssec] k2 < N2 ->              (k2' = k2+1);
  [pcssec] k2 < N2 ->              (k2' = k2+1);
endmodule

module PCS
  q: [0..N1-1] init 0;
  y: [0..2] init 0; // 0: idle, 1: busy with PU, 2: busy with SU
  [pripcs] y = 0            ->       (y' = 1);
  [pripcs] q < N1-1 & y = 1 ->       (q' = q+1);
  [priscs] y = 2            ->       (y' = 1);
  [priorb] y = 2            ->       (y' = 1);
  [secpcs] c = 1 & y = 0    ->       (y' = 2);
  [orbpcs] c = 1 & y = 0    ->       (y' = 2);
  [pcspri] y = 1 & q = 0    -> mu1 : (y' = 0);
  [pcspri] y = 1 & q > 0    -> mu1 : (q' = q-1);
  [pcssec] y = 2            -> mu1 : (y' = 0);
endmodule

module SCS
  c: [0..1] init 0; // 0: idle, 1: busy with SU
  [secscs] c = 0 ->       (c' = 1);
  [secorb] c = 1 ->       (c' = 0);
  [priscs] c = 0 ->       (c' = 1);
  [priorb] c = 1 ->       (c' = 0);
  [orbscs] c = 0 ->       (c' = 1);
  [scssec] c = 1 -> mu2 : (c' = 0);
endmodule

module Orbit
  o: [0..N2-1] init 0;
  [orbscs] o > 0             -> o*nu : (o' = o-1);
  [orbpcs] o > 0             -> o*nu : (o' = o-1);
  [secorb] o < N2-2 & y != 0 ->        (o' = o+2);
  [priorb] o < N2-2          ->        (o' = o+2);
```

```
endmodule

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

// average number of active primary users (= N1-M1)
rewards "K1"
  true : k1;
endrewards

// average number of jobs of primary users in network (= N1-K1)
rewards "M1"
  true : q + ( y = 1 ? 1 : 0 );
endrewards

// average number of active secondary users (= N2-M2)
rewards "K2"
  true : k2;
endrewards

// average number of jobs of secondary users in network (= N2-K2)
rewards "M2"
  true : o + c + ( y = 2 ? 1 : 0 );
endrewards

// utilization of PCS
rewards "U1"
  y != 0 : 1;
endrewards

// utilization of SCS
rewards "U2"
  c != 0 : 1;
endrewards

// orbit size
rewards "O"
  true : o;
endrewards

// ------------------------------------------------------------------
// end of model
// ------------------------------------------------------------------
// ------------------------------------------------------------------
// CognitiveRadio2.csl
// ------------------------------------------------------------------

"K1": R{"K1"}=? [ S ]
"T1": (N1-"K1")/(lambda1*"K1");

"K2": R{"K2"}=? [ S ]
"T2": (N2-"K2")/(lambda2*"K2");

"M1": R{"M1"}=? [ S ]
"K1M1": "K1"+"M1";

"M2": R{"M2"}=? [ S ]
"K2M2": "K2"+"M2";

"U1": R{"U1"}=? [ S ]
```

```
"U2": R{"U2"}=? [ S ]

"O": R{"O"}=? [ S ]
"N2O": "O"/N2
```

C A Cognitive Radio Network with Non-reliable Servers and Collisions

```
// ------------------------------------------------------------------
// CognitiveRadio1.prism
// Finite-source cognitive radio networks with collisions.
//
// This implements the model described in
//
//   Hamza Nemouchi, Janos Sztrik:
//   "Performance Simulation of Non-reliable Servers in
//   Finite-Source Cognitive Radio Networks with Collision"
//   Information Technologies and Mathematical Modelling -
//   Queueing Theory and Applications, Communications in
//   Computer and Information Science; Vol. 800 (2017) 194-203,
//   Springer Verlag
//
// (c) 2018, Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// parameters
// ------------------------------------------------------------------

const int N1; // = 20; // number of primary users (PUs)
const int N2; // = 50; // number of secondary users (SUs)

const double lambda1; // = 0.1; // generation rate of PUs
const double lambda2; // = 0.1; // generation rate of SUs

const double mu1; // = 1; // service intensity at primary channel service (PCS)
const double mu2; // = 1; // service intensity at secondary channel service (SCS)

const double nu; // = 20; // retrial intensity

const double gamma1; // = 0.05; // primary failure rate
const double gamma2; // = 0.05; // secondary failure rate

const double sigma1; // = 1; // primary repair rate
const double sigma2; // = 1; // secondary repair rate

const int pfail;  // = 1; // PCS may fail
const int sfail;  // = 1; // SCS may fail

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module Primary
  k1: [0..N1] init N1;
  [pripcs] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
```

```
  [priscs] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [priorb] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [pcspri] k1 < N1 ->              (k1' = k1+1);
endmodule

module Secondary
  k2: [0..N2] init N2;
  [secscs] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [secpcs] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [secorb] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [scssec] k2 < N2 ->              (k2' = k2+1);
  [pcssec] k2 < N2 ->              (k2' = k2+1);
endmodule

module PCS
  q:  [0..N1-1] init 0;
  y:  [0..2] init 0; // 0: idle, 1: busy with PU, 2: busy with SU
  pf: [0..1] init 0; // 0: working, 1: failed
  [pripcs] y = 0 & pf = 0                          ->          (y' = 1);
  [pripcs] q < N1-1 & y = 1 & pf = 0               ->          (q' = q+1);
  [priscs] y = 2 & pf = 0                          ->          (y' = 1);
  [priorb] y = 2  & pf = 0                         ->          (y' = 1);
  [secpcs] c = 1 & y = 0 & pf = 0                  ->          (y' = 2);
  [orbpcs] c = 1 & y = 0 & pf = 0                  ->          (y' = 2);
  [pcspri] y = 1 & q = 0 & pf = 0                  -> mu1    : (y' = 0);
  [pcspri] y = 1 & q > 0 & pf = 0                  -> mu1    : (q' = q-1);
  [pcssec] y = 2 & pf = 0                          -> mu1    : (y' = 0);
  [pfail1]  pfail = 1 & pf = 0 & y = 0             -> gamma1 : (pf' = 1);
  [pfail2]  pfail = 1 & pf = 0 & y = 1 & q < N1-1  -> gamma1 : (pf' = 1) & (y' = 0) & (q' = q+1);
  [pfail3]  pfail = 1 & pf = 0 & y = 2             -> gamma1 : (pf' = 1) & (y' = 0);
  [pfail4]  pfail = 1 & pf = 0 & y = 2             -> gamma1 : (pf' = 1) & (y' = 0);
  [prepair] pf = 1                                 -> sigma1 : (pf' = 0);
endmodule

module SCS
  c:  [0..1] init 0; // 0: idle, 1: busy with SU
  sf: [0..1] init 0; // 0: working, 1: failed
  [secscs]  c = 0 & sf = 0             ->          (c' = 1);
  [secorb]  c = 1 & sf = 0             ->          (c' = 0);
  [priscs]  c = 0 & sf = 0             ->          (c' = 1);
  [priorb]  c = 1 & sf = 0             ->          (c' = 0);
  [orbscs]  c = 0 & sf = 0             ->          (c' = 1);
  [scssec]  c = 1 & sf = 0             -> mu2    : (c' = 0);
  [pfail3]  c = 0 & sf = 0             ->          (c' = 1);
  [sfail1]  sfail = 1 & sf = 0 & c = 0 -> gamma2 : (sf' = 1);
  [sfail2]  sfail = 1 & sf = 0 & c = 1 -> gamma2 : (sf' = 1) & (c' = 0);
  [srepair] sf = 1                     -> sigma2 : (sf' = 0);
endmodule

module Orbit
  o: [0..N2-1] init 0;
  [orbscs] o > 0             -> o*nu : (o' = o-1);
  [orbpcs] o > 0             -> o*nu : (o' = o-1);
  [secorb] o < N2-2 & y != 0 ->        (o' = o+2);
  [priorb] o < N2-2          ->        (o' = o+2);
  [sfail2] o < N2-1          ->        (o' = o+1);
  [pfail4] o < N2-1 & c = 1  ->        (o' = o+1);
endmodule

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

```
// ------------------------------------------------------------------

// average number of active primary users (= N1-M1)
rewards "K1"
  true : k1;
endrewards

// average number of jobs of primary users in network (= N1-K1)
rewards "M1"
  true : q + ( y = 1 ? 1 : 0 );
endrewards

// average number of active secondary users (= N2-M2)
rewards "K2"
  true : k2;
endrewards

// average number of jobs of secondary users in network (= N2-K2)
rewards "M2"
  true : o + c + ( y = 2 ? 1 : 0 );
endrewards

// utilization of PCS
rewards "U1"
  y != 0 : 1;
endrewards

// utilization of SCS
rewards "U2"
  c != 0 : 1;
endrewards

// orbit size
rewards "O"
  true : o;
endrewards

// ------------------------------------------------------------------
// end of model
// ------------------------------------------------------------------
// ------------------------------------------------------------------
// CognitiveRadio1.csl
// ------------------------------------------------------------------

"K1": R{"K1"}=? [ S ]
"T1": (N1-"K1")/(lambda1*"K1");

"K2": R{"K2"}=? [ S ]
"T2": (N2-"K2")/(lambda2*"K2");

"M1": R{"M1"}=? [ S ]
"K1M1": "K1"+"M1";

"M2": R{"M2"}=? [ S ]
"K2M2": "K2"+"M2";

"U1": R{"U1"}=? [ S ]
"U2": R{"U2"}=? [ S ]

"O": R{"O"}=? [ S ]
"N2O": "O"/N2
```

D A Cognitive Communication System with Network Service Breakdown

```
// ------------------------------------------------------------------
// CognitiveSystem1.prism
// Finite-source cognitive infocommunication system.
//
// This implements the model described in
//
//   Attila Kuki, Tamas Berczes, Bela Almasi, Janos Sztrik:
//   "A queueing model to study the effect of network service breakdown
//   in a CogInfoCom system",CogInfoCom 2013, 4th IEEE International
//   Conference on Cognitive Infocommunications, December 2-5, 2013,
//   Budapest, Hungary.
//
// (c) 2018, Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// parameters
// ------------------------------------------------------------------

const int B;  // = 3;  // buffer size
const int N1; // = 3;  // number of intelligent entities
const int N2; // = 50; // number of normal entities

const double lambda1; // = 0.3; // generation rate of intelligent entities
const double lambda2; // = 1.2; // generation rate of normal entities

const double mu1; // = 1; // service rate
const double mu2; // = 1; // service rate in limited state

const double nu; // = 4; // retrial rate

const double p;  // = 0.5; // probability server state changes from level 1 to 2

const double delta; // server's failure rate
const double beta;  // server's repair rate

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module Intelligent
  k1: [0..N1] init N1;
  [intser] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [intorb] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [serint] k1 < N1 ->              (k1' = k1+1);
endmodule

module Normal
  k2: [0..N2] init N2;
  [norser] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [sernor] k2 < N2 ->              (k2' = k2+1);
endmodule

module Orbit
```

```
  o: [0..N1] init 0;
  [intorb] o < N1 & c > 0 & q1+q2 = B ->       (o' = o+1);
  [orbser] o > 0                      -> o*nu: (o' = o-1);
endmodule

formula mu = (y = 1 ? mu1 : mu2);

module Server
  q1: [0..N1] init 0;
  q2: [0..N2] init 0;
  y: [1..3] init 1; // 1: normal, 2: limited, 3: failed
  c: [0..2] init 0; // 0: idle, 1: busy with intelligent, 2: busy with normal
  [intser] c = 0                     -> (c' = 1);
  [intser] c > 0 & q1+q2 < min(B,N1) -> (q1' = q1+1);
  [orbser] c = 0                     -> (c' = 1);
  [orbser] c > 0 & q1+q2 < min(B,N1) -> (q1' = q1+1);
  [norser] c = 0                     -> (c' = 2);
  [norser] c > 0 & q1+q2 < min(B,N2) -> (q2' = q2+1);
  [serint] c = 1 & q1 = 0 & q2 = 0 & y < 3 -> mu : (c' = 0);
  [serint] c = 1 & q1 > 0 & y < 3          -> mu : (c' = 1) & (q1' = q1-1);
  [serint] c = 1 & q2 > 0 & y < 3          -> mu : (c' = 2) & (q2' = q2-1);
  [sernor] c = 2 & q1 = 0 & q2 = 0 & y < 3 -> mu : (c' = 0);
  [sernor] c = 2 & q1 > 0 & y < 3          -> mu : (c' = 1) & (q1' = q1-1);
  [sernor] c = 2 & q2 > 0 & y < 3          -> mu : (c' = 2) & (q2' = q2-1);
  [fail12] y = 1 -> delta*p :     (y' = 2);
  [fail13] y = 1 -> delta*(1-p) : (y' = 3);
  [fail23] y = 2 -> delta :       (y' = 3);
  [repair] y > 1 -> beta :        (y' = 1);
endmodule

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

// orbit size
rewards "O"
  true : o;
endrewards

// probability server is in limited state
rewards "Pl"
  y = 2 : 1;
endrewards

// probability server is broken
rewards "Pb"
  y = 3 : 1;
endrewards

// average number of intelligent entities in source
rewards "K1"
  true : k1;
endrewards

// average number of normal entities in source
rewards "K2"
  true : k2;
endrewards

// number of intelligent entities in queue
rewards "Q1"
```

```
  true : q1;
endrewards

// number of normal entities in queue
rewards "Q2"
  true : q2;
endrewards

// ------------------------------------------------------------------
// end of model
// ------------------------------------------------------------------

// ------------------------------------------------------------------
// CognitiveSystem1.csl
// ------------------------------------------------------------------

"O": R{"O"}=? [ S ];
"Pl": R{"Pl"}=? [ S ];
"Pb": R{"Pb"}=? [ S ];
"K1": R{"K1"}=? [ S ];
"K2": R{"K2"}=? [ S ];
"T1": (N1-"K1")/(lambda1*"K1");
"T2": (N2-"K2")/(lambda2*"K2");
"Q1": R{"Q1"}=? [ S ];
"Q2": R{"Q2"}=? [ S ];
"W1": ("Q1"+"O")/(lambda1*"K1");
"W2": "Q2"/(lambda2*"K2");
"WA1": "T1"-"W1"-(1/mu1);
"WA2": "T2"-"W2"-(1/mu2);
```

E A Multilayered Cognitive Communication System with Network Series Breakdown

```
// ------------------------------------------------------------------
// CognitiveSystem2.prism
// Finite-source multi-layered cognitive infocommunication system.
//
// This implements the model described in
//
//   Attila Kuki, Tamas Berczes, Bela Almasi, Janos Sztrik:
//   "Investigating the effect of network service breakdown in
//   multilayered cognitive communication system", CogInfoCom 2014,
//   5th IEEE International Conference on Cognitive Infocommunications,
//   Vietri sul Mare, Italy, 2014.
//
// (c) 2018, Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
// Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// parameters
// ------------------------------------------------------------------

const int B;  // = 3;  // buffer size
const int N1; // = 3;  // number of intelligent entities
```

```
const int N2; // = 50; // number of normal entities

const double lambda1; // = 0.3; // generation rate of intelligent entities
const double lambda2; // = 1.2; // generation rate of normal entities

const double mu1; // = 1; // rate of server 1
const double mu2; // = 1; // rate of server 2
const double mu3; // = 1; // rate of server 3

const double nu; // = 4; // retrial rate

const double delta; // server's failure rate
const double beta;  // server's repair rate

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module Intelligent
  k1: [0..N1] init N1;
  [intser] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [intorb] k1 > 0  -> k1*lambda1 : (k1' = k1-1);
  [serint] k1 < N1 ->              (k1' = k1+1);
endmodule

module Normal
  k2: [0..N2] init N2;
  [norser] k2 > 0  -> k2*lambda2 : (k2' = k2-1);
  [sernor] k2 < N2 ->              (k2' = k2+1);
endmodule

module Orbit
  o: [0..N1] init 0;
  [intorb] o < N1 & !(stage1 & c1 = 0) & q1+q2 = min(B,N1) ->        (o' = o+1);
  [orbser] o > 0                                           -> o*nu : (o' = o-1);
endmodule

formula stage1 = y1 = 1 & c2 = 0 & c3 = 0;
formula stage2 = y2 = 1;
formula stage3 = y3 = 1;

module ServiceUnit
  q1: [0..N1] init 0;
  q2: [0..N2] init 0;
  c1: [0..2] init 0; // 0: idle, 1: busy with intelligent, 2: busy with normal
  c2: [0..2] init 0; // 0: idle, 1: busy with intelligent, 2: busy with normal
  c3: [0..2] init 0; // 0: idle, 1: busy with intelligent, 2: busy with normal
  y1: [1..2] init 1; // 1: normal, 2: failed
  y2: [1..2] init 1; // 1: normal, 2: failed
  y3: [1..2] init 1; // 1: normal, 2: failed

  [intser] stage1 & c1 = 0 -> (c1' = 1);
  [intser] !(stage1 & c1 = 0) & q1+q2 < min(B,N1) -> (q1' = q1+1);
  [orbser] stage1 & c1 = 0 -> (c1' = 1);
  [orbser] !(stage1 & c1 = 0) & q1+q2 < min(B,N1) -> (q1' = q1+1);
  [norser] stage1 & c1 = 0 -> (c1' = 2);
  [norser] !(stage1 & c1 = 0) & q1+q2 < min(B,N2) -> (q2' = q2+1);

  [intser2] stage1 & c1 = 1 & q1 = 0 & q2 = 0 -> mu1 : (c1' = 0) & (c2' = 1);
  [intser2] stage1 & c1 = 1 & q1 > 0          -> mu1 : (c1' = 1) & (c2' = 1) & (q1' = q1-1);
  [intser2] stage1 & c1 = 1 & q2 > 0          -> mu1 : (c1' = 2) & (c2' = 1) & (q2' = q2-1);
```

```
  [norser2] stage1 & c1 = 2 & q1 = 0 & q2 = 0 -> mu1 : (c1' = 0) & (c2' = 2);
  [norser2] stage1 & c1 = 2 & q1 > 0          -> mu1 : (c1' = 1) & (c2' = 2) & (q1' = q1-1);
  [norser2] stage1 & c1 = 2 & q2 > 0          -> mu1 : (c1' = 2) & (c2' = 2) & (q2' = q2-1);

  [intser3] stage2 & c2 = 1 -> mu2 : (c2' = 0) & (c3' = 1);
  [norser3] stage2 & c2 = 2 -> mu2 : (c2' = 0) & (c3' = 2);

  [serint] stage3 & c3 = 1 -> mu3 : (c3' = 0);
  [sernor] stage3 & c3 = 2 -> mu3 : (c3' = 0);

  [fail11] y1 = 1 -> delta : (y1' = 2);
  [fail12] y2 = 1 -> delta : (y2' = 2);
  [fail13] y3 = 1 -> delta : (y3' = 2);

  [repair1] y1 = 2 -> beta : (y1' = 1);
  [repair2] y2 = 2 -> beta : (y2' = 1);
  [repair3] y3 = 2 -> beta : (y3' = 1);
endmodule

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

// orbit size
rewards "O"
  true : o;
endrewards

// average number of intelligent entities in source
rewards "K1"
  true : k1;
endrewards

// average number of normal entities in source
rewards "K2"
  true : k2;
endrewards

// number of intelligent entities in queue
rewards "Q1"
  true : q1;
endrewards

// number of normal entities in queue
rewards "Q2"
  true : q2;
endrewards

// ------------------------------------------------------------------
// end of model
// ------------------------------------------------------------------
```

```
// ------------------------------------------------------------------
// CognitiveSystem1.csl
// ------------------------------------------------------------------

"O": R{"O"}=? [ S ];
"K1": R{"K1"}=? [ S ];
"K2": R{"K2"}=? [ S ];
"T1": (N1-"K1")/(lambda1*"K1");
"T2": (N2-"K2")/(lambda2*"K2");
"Q1": R{"Q1"}=? [ S ];
```

```
"Q2": R{"Q2"}=? [ S ];
"Q":  "Q1"+"Q2";
"W1": ("Q1"+"O")/(lambda1*"K1");
"W2": "Q2"/(lambda2*"K2");
"WA1": "T1"-"W1"-(1/mu1);
"WA2": "T2"-"W2"-(1/mu2);
```