MODELING FINITE-SOURCE RETRIAL QUEUEING SYSTEMS WITH UNRELIABLE HETEROGENEOUS SERVERS AND DIFFERENT SERVICE POLICIES USING MOSEL

Patrick Wüchner, Hermann de Meer,
Faculty of Informatics and Mathematics,
University of Passau,
Innstraße 43,
94032 Passau, Germany.
patrick.wuechner@uni-passau.de

Gunter Bolch,
Institute of Computer Science,
University of Erlangen,
Martensstraße 3,
91508 Erlangen, Germany.
bolch@informatik.uni-erlangen.de

János Roszik, János Sztrik,
Faculty of Informatics,
University of Debrecen,
Egyetem tér 1. Po.Box 12,
4010 Debrecen, Hungary.
jsztrik@inf.unideb.hu

KEYWORDS
Performance and Reliability Evaluation, Retrial Queuing System Model, Unreliable Heterogeneous Servers.

ABSTRACT

This paper deals with the performance analysis of multiple server retrial queueing systems with a finite number of homogeneous sources of calls, where the heterogeneous servers are subject to random breakdowns and repairs. The requests are serviced according to Random Selection and Fastest Free Server disciplines.

The novelty of this investigation is the introduction of different service rates and different service policies together with the unreliability of the servers, which has essential influence on the performance of the system, and thus, it plays an important role in practical modeling of computer and communication systems. All random variables involved in the model construction are assumed to be exponentially distributed and independent of each other.

The main steady-state performability measures are derived, and several numerical calculations are carried out by the help of the MOSEL tool (Modeling, Specification and Evaluation Language) under different service disciplines. The numerical results are graphically displayed to illustrate the effect of failure rates on the mean response time and on the overall system utilization.

I. INTRODUCTION

Retrial queueing systems (also known as queueing systems with repeated attempts or with returning customers) are characterized by the following feature: a primary request finding all servers busy on arrival does not wait in a queue, but leaves the service area and after some random time repeats its demand. For the most important results on this type of queues see, for example (Falin and Templeton 1997), (Artalejo 1998), (Artalejo 1999), and (Falin 1999). This feature plays a special role in many computer and communication systems having a significant negative impact on the performance characteristics of the system. For some examples in the field of computer systems and communication networks see (Li and Yang 1995), (Janssens 1997), and (Tran-Gia and Mandjes 1997).

In general, the components of practical computer systems are subject to random breakdowns. This has a heavy influence on the performance measures just as the retrial phenomenon. Thus, if we model computer systems containing unreliable components it is important to take it into account in the model construction. Of course, the breakdown of the servers has the most significant negative impact on the performance of the most frequently used client-server architecture. For modeling this type of systems, both infinite and finite-source retrial queues with server breakdowns were applied—see, for example, (Artalejo 1994), (Aissani and Artalejo 1998), (Wang et al. 2001), and (Almasi et al. 2005). Queueing systems with heterogeneous servers are still an interesting topic. For recent results confer, for example, (Rykov 2001), (Nobel and Tijms 2000). However, for retrial queueing systems with heterogeneous servers we have found only (Pourbabai 1987). To the best knowledge of the authors there is no paper on finite-source retrial queues with heterogeneous servers, not even in purely reliable case.

In this paper, we analyze the finite-source retrial queue with unreliable heterogeneous (asymmetric) servers, that is, the servers have different parameters in service, failure, and repair rates. In the present study, the most important heterogeneous characteristic is the service rate, since we compare two service policies, namely Random Service (RS) and Fastest Free Server (FFS). In the case of RS discipline, the requests are assigned to the idle servers randomly, and in the FFS case, the requests are assigned to the fastest available free server.

The purpose of this paper is to generalize the models of (Falin 1999) and (Almasi et al. 2005). The novelty of this investigation is the introduction of different service rates and different service policies together with the unreliability of the servers.

The main steady-state performability measures are derived, and several numerical calculations are carried out by the help of the MOSEL tool (Begain et al. 2001) under different service disciplines. The numerical results are graphically displayed to illustrate the effect of failure rates on the mean response time and on the overall system utilization.

The organization of the paper is as follows. In the next section, we give the mathematical model description and derive the performance measures. Then, we use the efficient software tool MOSEL to formulate the model

Print 0-9553018-4-X, CD 0-9553018-3-1

and to obtain the performance measures. In Section 3, we present some numerical examples for the models under different service disciplines. The results are graphically displayed using the IGL (Intermediate Graphical Language) interpreter which belongs to MOSEL. By the help of these figures we illustrate the effect of failure rates on the mean response time and on the overall system utilization. Section 4 is devoted to some conclusions.

II. THE $M/\vec{M}/c//K$ RETRIAL QUEUEING MODEL WITH UNRELIABLE SERVERS AND DIFFERENT SERVICE POLICIES

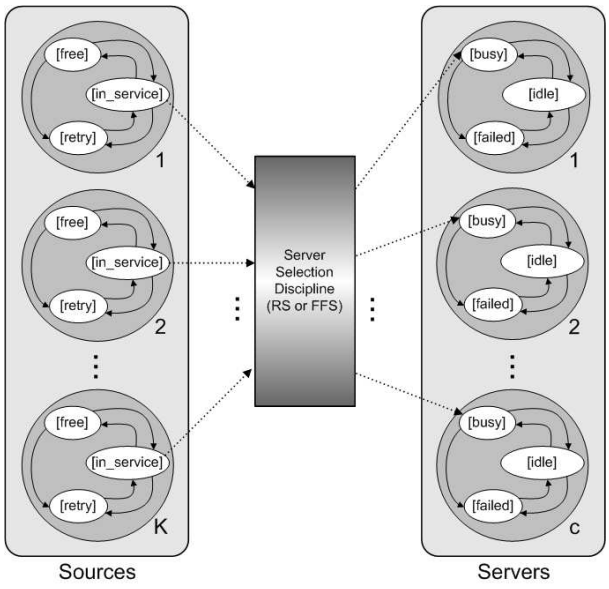

Fig. 1. Finite-source retrial queueing system

Consider a finite-source retrial queueing system with c servers, where the primary calls are generated by K, $c \leq K < \infty$, sources (see Fig. 1). Each server can be in operational (`up`) or non-operational (`failed`) states, and if it is up it can be `idle` or `busy`. Each source can be in three states: generating a primary call (`free`), sending repeated calls (`retry`), and under service (`in_service`) by one of the servers. If a source is free at time t, it can generate a primary call during the interval $(t, t+dt)$ with probability $\lambda dt + o(dt)$. If one of the servers is up and idle at the moment of the arrival of the call then the service of the call starts. In the case of Random Service (RS) discipline, the requests are assigned to the available free servers randomly, but in the Fastest Free Server (FFS) case the availability and idleness of the servers are always examined according to the highest service rates. The service is finished during the interval $(t, t+dt)$ with probability $\mu_k dt + o(dt)$ if the kth server is available.

Server k can fail during the interval $(t, t+dt)$ with probability $\delta_k dt + o(dt)$ if it is idle, and with probability $\gamma_k dt + o(dt)$ if it is busy. If the server fails in busy state, the interrupted request returns to the orbit and, thus, the respective source will retry to get service. If $\delta_k = 0$ and $\gamma_k > 0$, or $\delta_k = \gamma_k > 0$, *active* or *independent breakdowns* can be discussed, respectively. The repairman follows FIFO discipline to fix up the server breakdowns, the repair time of the kth server is exponentially distributed with a finite mean $1/\tau_k$. If all the servers are failed, two different cases can be treated: In the *blocked sources* case, all the operations are stopped except from the repair of the servers. On the other hand, in the *unblocked (intelligent) sources* case, only service is interrupted but all the other operations are continued. If all the servers are busy or failed at the moment of the arrival of a call, the source starts generating a Poisson flow of retrial calls with rate ν until it finds an available free server. After service, the source can generate a new primary call, and the server becomes idle so it can serve a new call. All the times involved in the model construction are assumed to be mutually independent of each other.

The state of the system at time t can be described by the process $X(t) = (\alpha_1(t), \ldots, \alpha_c(t); N(t))$, where $N(t) \leq K$ is the number of retrying sources at time t, and $\alpha_k(t), k = 1, \ldots, c$, denotes the state of the kth server at time t. If there is a customer under service at the kth server, we define $\alpha_k(t) = 1$, if it is operational and idle then $\alpha_k(t) = 0$, otherwise the server is failed and $\alpha_k(t) = -1$.

Because of the exponentiality of the involved random variables and the finite number of sources, this process is a Markov chain with a finite state space. Since the state space of the process $(X(t), t \geq 0)$ is finite, the process is ergodic for all reasonable values of the rates involved in the model construction. From now on, we assume that the system is in the steady-state. Because of the fact that the state space of the describing Markov chain is very large, it is difficult to calculate the system measures in the traditional way of solving the system of steady-state global balance equations. To simplify this procedure we use the software tool MOSEL.

Let us define the stationary probabilities by:

$$P(s_1, \ldots, s_c, j) = \lim_{t \to \infty} P\{\alpha_1(t) = s_1, \ldots, \alpha_c(t) = s_c, N(t) = j\},$$

$$s_1, \ldots, s_c = -1, 0, 1, \;\; j = 0, \ldots, K^*,$$

where K^* is the number of sources in service and given by:

$$K^* = K - \sum_{s_k, s_k = 1} s_k.$$

Furthermore, let us denote by $N(\infty)$ the number of retrying sources, $C(\infty)$ the number of busy servers, $A(\infty)$ the number of available servers at steady-state, and denote by $p_{kj} = P\{C(\infty) = k, N(\infty) = j\}$ the joint steady-state distribution of the number of busy servers and the number of retrying sources.

Once we have obtained the above defined probabilities, the main steady-state system performance measures can be derived as follows:

- *Mean number of retrying sources (calls in orbit):*

$$N = E[N(\infty)] = \sum_{k=0}^{c}\sum_{j=1}^{K} jp_{kj} = \sum_{s_1,\ldots,s_c}\sum_{j=1}^{K^*} jP(s_1,\ldots,s_c,j).$$

- *Utilization of the kth server:*

$$U_k = \sum_{s_1,\ldots,s_c,s_k=1}\sum_{j=0}^{K^*} P(s_1,\ldots,s_c,j), \quad k=1,\ldots,c.$$

- *Mean number of busy servers:*

$$C = E[C(\infty)] = \sum_{k=1}^{c} U_k.$$

- *Mean number of calls staying in the orbit or in service:*

$$M = E[N(\infty) + C(\infty)] = N + C.$$

- *Utilization of the repairman:*

$$U_R = \sum_{\substack{s_1,\ldots,s_c \\ -1\in\{s_1,\ldots,s_c\}}}\sum_{j=0}^{K^*} P(s_1,\ldots,s_c,j).$$

- *Utilization of the sources:*

$$U_S = \begin{cases} \frac{E[K-C(\infty)-N(\infty);A(\infty)>0]}{K} & \text{(blocked case)}, \\ \frac{E[K-C(\infty)-N(\infty)]}{K} & \text{(unblocked case)}. \end{cases}$$

- *Overall utilization of the system:*

$$U_O = C + KU_S + U_R.$$

- *Mean rate of generation of primary calls:*

$$\overline{\lambda} = \begin{cases} \lambda E[K-C(\infty)-N(\infty);A(\infty)>0] & \text{(blocked case)}, \\ \lambda E[K-C(\infty)-N(\infty)] & \text{(unblocked case)}. \end{cases}$$

- *Mean waiting time:* $E[W] = N/\overline{\lambda}$.
- *Mean response time:* $E[T] = M/\overline{\lambda}$.

Validation of Results

The numerical results generated by the MOSEL tool in the reliable case (see model descriptions in Appendix) were validated by the Pascal program given in the book of Falin and Templeton (Falin and Templeton 1997). The service rates are the same for all servers in each case. In Table 1 we can see that the corresponding performance measures with RS and FFS disciplines are very close to the reliable case and to each other with very low failure and very high repair rates. The results are the same up to the 6th decimal digit.

The MOSEL models were tested in unreliable case, too. Since only the unreliable single server case was treated earlier, the results were validated by the $M/M/1//K$ retrial model with server breakdowns which was studied in (Almasi et al. 2005). The numerical calculations given in (Almasi et al. 2005) correspond to the examples of the paper at hand.

III. NUMERICAL EXAMPLES

In this section, we present some numerical results to illustrate graphically the differences between the service disciplines in the mean response time, in the utilization of the servers and in the overall system utilization. In the legends of the figures, the FFS policy is referred to as *ordered*, and the random case where the service rate of the servers is the average of the rates of the heterogeneous cases is referred to as *averaged random*. In all cases we consider independent breakdowns with the same failure and repair rate for all servers, respectively.

The input system parameters of the Figures 2, 3, and 4 are collected in Table 2 where the RS and FFS disciplines are compared.

Table 2: Input system parameters of Figs. 2, 3, and 4

	c	K	λ	$\mu_1,\ldots,\mu_c-\mu_{avg}$	ν	δ,γ	τ
Fig. 2, 3, 4	4	20	1	8, 5, 4, 1 – 4.5	4	x axis	0.2

The system parameters of Figures 5, 6, 7, 8, and 9 are collected in Table 3 where only the FFS discipline is treated.

Table 3: Input system parameters of Figs. 5, 6, 7, 8, and 9

	c	K	λ	μ_1,μ_2	ν	δ,γ	τ
Fig. 5, 6	2	5	0.2	1, 1	1.1	x axis	0.01
Fig. 7, 8, 9	2	5	0.2	1.5, 0.5	1.1	x axis	0.01

In Figures 5 and 7, the effects of the server failure rate on the mean response time are displayed with homogeneous and different servers, respectively. In Figures 6 and 9, we can see the effect of the server failure rate on the overall utilization. In Figure 8, the server utilization is shown in the case of different servers. In each figure, the reliable, the blocked, and unblocked (intelligent) cases are analyzed .

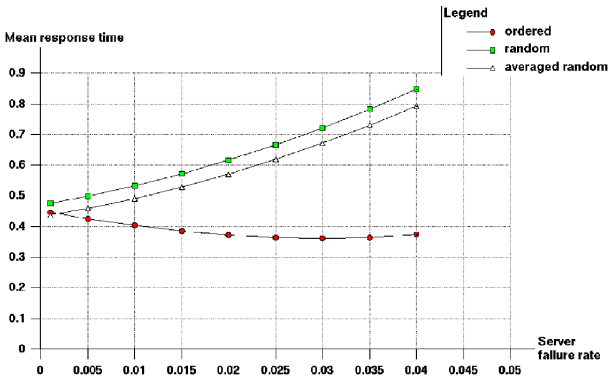

Fig. 2. Mean response time versus server failure rate

Discussion of Results

- In Figure 2, it is shown how the increase of the server failure rate affects the mean response time. The averaged random case has a little better response time than the not averaged random case comparable to the former figures. The surprising decrease in the mean response time in FFS case can be explained by the help of Figure 3.

Table 1: Validations in the reliable case

	Pascal (Falin and Templeton 1997)	RS	FFS
Number of servers:	4	4	4
Number of sources:	20	20	20
Request generation rate:	0.1	0.1	0.1
Service rate:	1	1	1
Retrial rate:	1.2	1.2	1.2
Server failure rate:	–	1e-25	1e-25
Server repair rate:	–	1e+25	1e+25
Mean waiting time:	0.1064954794	0.1064959317	0.1064959929
Mean number of busy servers:	1.8007480431	1.8007485102	1.8007485548
Mean number of call-repeating sources:	0.1917715262	0.1917717923	0.1917718470

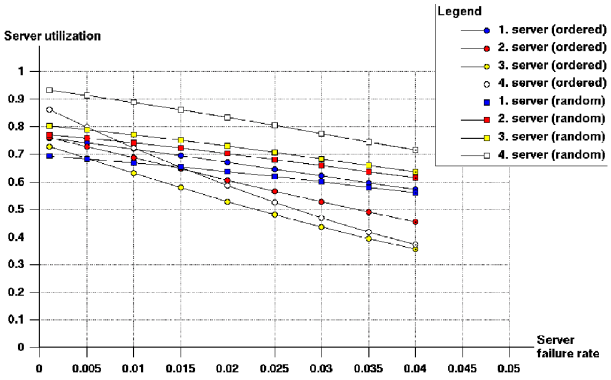

Fig. 3. Server utilization versus server failure rate

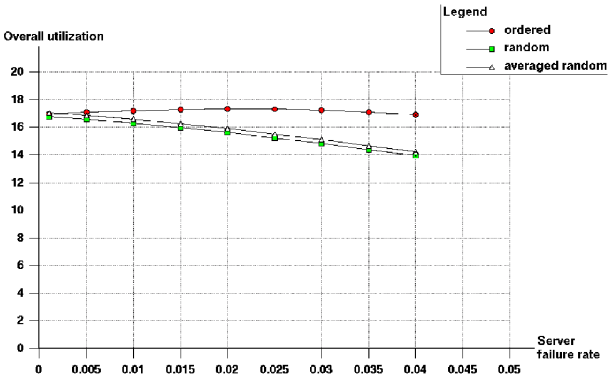

Fig. 4. Overall utilization versus server failure rate

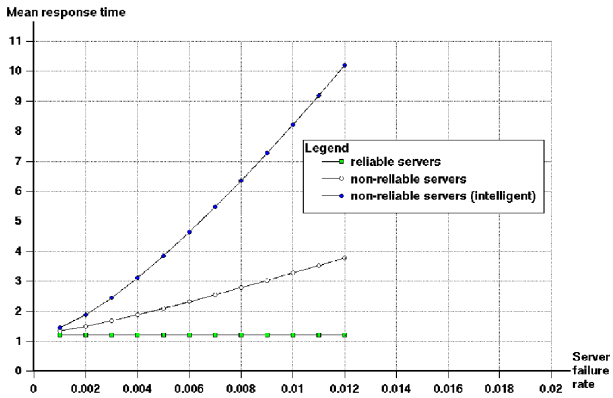

Fig. 5. $E[T]$ versus server failure rate

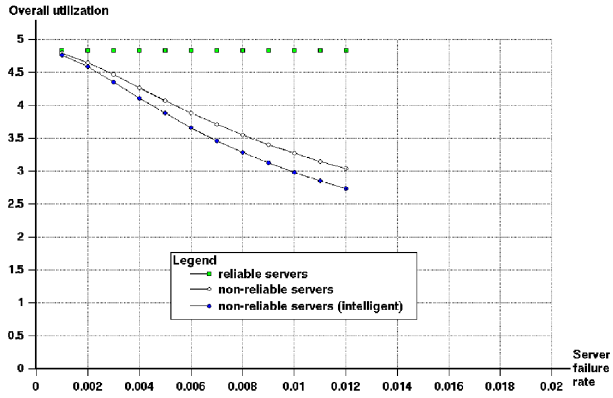

Fig. 6. U_O versus server failure rate

• In Figure 3, we can see the server utilization versus the server failure rate with the same parameter setup as in Figure 2. In RS case, the slowest server has the highest utilization and the fastest has the lowest, since it services the request much faster and the requests are assigned to the available and free servers with the same probability. In the beginning of the ordered (FFS) case, the slowest server has the highest utilization too, but as it fails more often, its service is interrupted more often and looses from its utilization much faster than the faster servers, since it gets requests to serve only if all the other servers busy or failed.

• In Figure 4, the overall utilization is displayed versus the server failure rate. Like the mean response time in Figure 2, the overall utilization is getting better for a while in the FFS case as the server failure rate increases.

• In Figures 5 and 7, it can be observed that the increase of the server failure rate can have a heavy impact on the mean response time, and as it increases the difference between the two unreliable model increases significantly.

• In Figures 6 and 9, it is shown that the overall utilization can be very low if the server failure rate increases and the repair rate is not high enough.

IV. CONCLUSIONS

In this paper, the performance of finite-source retrial queueing systems with homogeneous sources and unreliable heterogeneous (asymmetric) servers is studied. The novelty of the investigation is the introduction of different service rates and different service policies with the unreliability of the servers. The MOSEL software package was used to formulate the model and to calculate the steady-state system performance measures which were graphically displayed to show the differences between the service disciplines in the mean response time, in the utiliza-

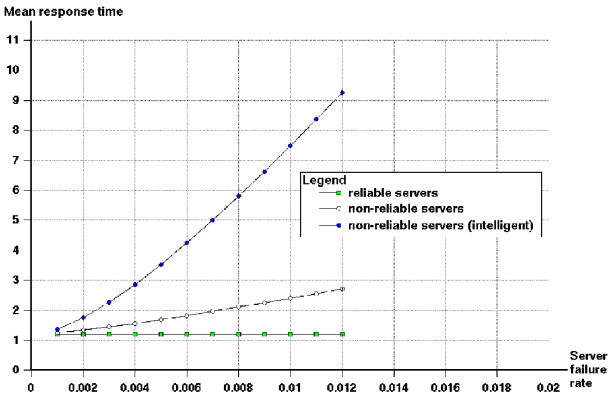

Fig. 7. $E[T]$ versus server failure rate

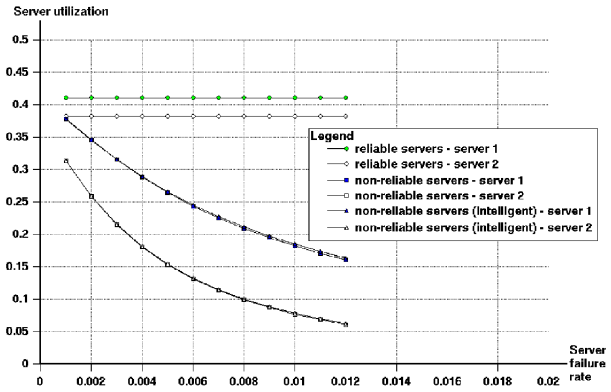

Fig. 8. Server utilization versus server failure rate

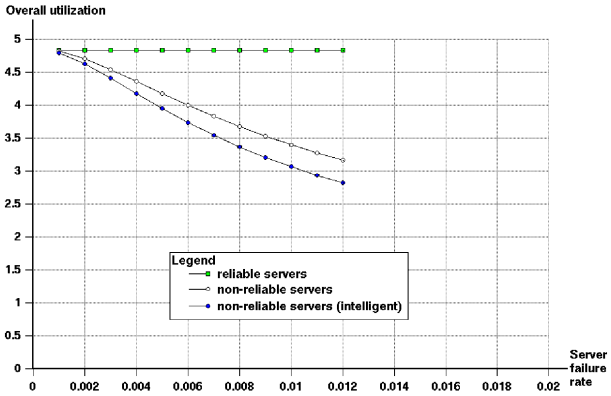

Fig. 9. U_O versus server failure rate

tion of the servers and in the overall system utilization versus server failure rate. It was demonstrated that the FFS discipline has more favorable system measures, as it was expected, and in the case of RS policy, it is more worthy to apply homogeneous servers with the average service rates of the heterogeneous case, at least with these setup of the parameters.

In future, we will extend our model to have a closer look on the effect of non-exponentially distributed service and retrial times. For this, we will employ the successor of MOSEL, called MOSEL-2 (Wuechner et al. 2006; Bolch et al. 2006), which is capable of specifying and evaluating non-Markovian discrete-event system models.

V. ACKNOWLEDGMENTS

This research is partially supported by the German-Hungarian Intergovernmental Scientific Cooperation, HAS-DFG, 436 UNG 113/180/0-1, by the Hungarian Scientific Research Fund, OTKA K60698/2006, by the Network of Excellence EuroFGI – IST 028022, and by EPSRC, GR/S69009/01.

REFERENCES

Aissani, A. and J. Artalejo (1998) "On the single server retrial queue subject to breakdowns. ", *Queueing Systems 30*, 309–321.

Almasi, B., J. Roszik, and J. Sztrik (2005) "Homogeneous finite-source retrial queues with server subject to breakdowns and repairs. ", *Mathematical and Computer Modelling 42*, 673–682.

Artalejo, J. (1994) "New results in retrial queueing systems with breakdown of the servers. ", *Statistica Neerlandica 48*, 23–36.

Artalejo, J. R. (1998) "Retrial queues with a finite number of sources. ", *J. Korean Math. Soc. 35*, 503–525.

Artalejo, J. R. (1999) "Accessible bibliography on retrial queues. ", *Mathematical and Computer Modelling 30*, 1–6.

Begain, K., G. Bolch, and H. Herold (2001). *Practical Performance Modeling – Application of the MOSEL Language*. Kluwer Academic Publishers.

Bolch, G., S. Greiner, H. de Meer, and K. Trivedi (2006). *Queueing Networks and Markov Chains* (2 ed.). New York: John Wiley & Sons.

Falin, G. and J. Templeton (1997). *Retrial Queues*. Chapman & Hall.

Falin, G. I. (1999) "A multiserver retrial queue with a finite number of sources of primary calls. ", *Mathematical and Computer Modelling 30*, 33–49.

Janssens, G. K. (1997) "The quasi-random input queueing system with repeated attempts as a model for collision-avoidance star local area network. ", *IEEE Transactions on Communications 45*, 360–364.

Li, H. and T. Yang (1995) "A single server retrial queue with server vacations and a finite number of input sources. ", *European Journal of Operational Research 85*, 149–160.

Nobel, R. D. and H. C. Tijms (2000) "Optimal control of a queueing system with heterogeneous servers and setup costs. ", *IEEE Trans. Autom. Control 45*, 780–794.

Pourbabai, B. (1987) "Markovian queueing systems with retrials and heterogeneous servers. ", *Computers and Mathematics with Applications 13*, 917–923.

Rykov, V. (2001) "Monotone control of queueing systems with heterogeneous servers. ", *Queueing Systems 37*, 391–403.

Tran-Gia, P. and M. Mandjes (1997) "Modeling of customer retrial phenomenon in cellular mobile networks. ", *IEEE Journal of Selected Areas in Communications 15*, 1406–1414.

Wang, J., J. Cao, and Q. Li (2001) "Reliability analysis of the retrial queue with server breakdowns and repairs. ", *Queueing Systems 38*, 363–380.

Wuechner, P., H. De Meer, J. Barner, and G. Bolch (2006) "A brief introduction to MOSEL-2. ", In R. German and A. Heindl (Eds.), *Proc. of MMB 2006 Conference*, pp. 473–476. GI/ITG/MMB, University of Erlangen: VDE Verlag.

AUTHOR BIOGRAPHIES

PATRICK WÜCHNER received his computer science Diploma in 2004 from the University of Erlangen-Nuernberg, Germany. Since then, he is research fellow and PhD student at the University of Passau, Germany. He is interested in research on mathematical performance and reliability modeling of computer systems, communication networks, and self-organizing systems.

HERMANN DE MEER is currently appointed as Full Professor at the University of Passau, Germany, and as Honorary Professor at University College London, UK. He is director of the Institute of IT-Security and Secu-

rity Law (ISL) at the University of Passau, Germany. He had been an Assistant Professor at Hamburg University, Germany, a Visiting Professor at Columbia University in New York City, USA, and a Reader at University College London, UK. His research interests include peer-to-peer systems, quality of service, Internet protocols, home networking, IT security, and mobile computing.

GUNTER BOLCH is Acad. Director the Department of Computer Science 4 (Operating Systems) at the University of Erlangen-Nuernberg, Germany. He studied telecommunication at the Technical Universities of Karlsruhe and Berlin and was Assistant Professor at the Department of Process Control at the University of Karlsruhe. After receiving his Ph.D. in 1973, he went to the University of Erlangen where from 1982 until his retirement in 2006 he was head of the Performance Modeling and Process Control research group at the Department of Computer Science 4. He was researching and lecturing in the area of performance modeling, process control, and operating systems.

JÁNOS ROSZIK received his MSc Degree in Computer Science in 2003 at the University of Debrecen, Hungary. He is currently a PhD student at the Department of Informatics Systems and Networks of the same university. His primary research interests are performance analysis of retrial queues and their application in modeling of telecommunication systems.

JÁNOS SZTRIK is a Full Professor at the Faculty of Informatics, University of Debrecen, Hungary. He obtained the Candidate of Mathematical Sciences Degree in Probability Theory and Mathematical Statistics in 1989 from the Kiev State University, Kiev, USSR, Habilitation from University of Debrecen in 1999, Doctor of the Hungarian Academy of Sciences, Budapest, 2002. His research interests are in the field of production systems modeling and analysis, queueing theory, reliability theory, and computer science.

APPENDIX

A. The MOSEL Model for the Random Service Policy

```
// ============================== Constant definitions ====
#define NT 20
#define NS 4
// ====================== Variables (input parameters) ====
VAR double prgen;
VAR double prretr;
<1..NS> VAR double prrun#;
<1..NS> VAR double cpubreak_idle#;
<1..NS> VAR double cpubreak_busy#;
<1..NS> VAR double cpurepair#;
// ================================== Node definitions ====
enum cpu_states {cpu_busy, cpu_idle, cpu_failed};
NODE busy_terminals[NT] = NT;
NODE retrying_terminals[NT] = 0;
NODE waiting_terminals[NS] = 0;
<1..NS> NODE cpu#[cpu_states] = cpu_idle;
        NODE freecpus[NS] = NS;
        NODE failedcpus[NS] = 0;
<1..NS> NODE sr#[NS] = 0;
// ======================================= Transitions ====
FROM cpu1[cpu_idle], busy_terminals, freecpus
        TO cpu1[cpu_busy], waiting_terminals
        W prgen*busy_terminals/freecpus;
FROM cpu2[cpu_idle], busy_terminals, freecpus
        TO cpu2[cpu_busy], waiting_terminals
        W prgen*busy_terminals/freecpus;
FROM cpu3[cpu_idle], busy_terminals, freecpus
        TO cpu3[cpu_busy], waiting_terminals
        W prgen*busy_terminals/freecpus;
FROM cpu4[cpu_idle], busy_terminals, freecpus
        TO cpu4[cpu_busy], waiting_terminals
        W prgen*busy_terminals/freecpus;
FROM busy_terminals TO retrying_terminals
        IF freecpus==0 W prgen*busy_terminals;
FROM cpu1[cpu_idle], retrying_terminals, freecpus
        TO cpu1[cpu_busy], waiting_terminals
        W prretr*retrying_terminals/freecpus;
FROM cpu2[cpu_idle], retrying_terminals, freecpus
        TO cpu2[cpu_busy], waiting_terminals
        W prretr*retrying_terminals/freecpus;
FROM cpu3[cpu_idle], retrying_terminals, freecpus
        TO cpu3[cpu_busy], waiting_terminals
        W prretr*retrying_terminals/freecpus;
FROM cpu4[cpu_idle], retrying_terminals, freecpus
        TO cpu4[cpu_busy], waiting_terminals
        W prretr*retrying_terminals/freecpus;
<1..NS><NS> FROM cpu<#1>[cpu_busy], waiting_terminals{
  TO cpu<#1>[cpu_idle], busy_terminals, freecpus W prrun<#1>;
  TO cpu<#1>[cpu_failed],retrying_terminals, failedcpus,sr<#2>(<#1>)
  W cpubreak_busy<#1>; }
<1..NS><NS> FROM cpu<#1>[cpu_idle], freecpus
  TO cpu<#1>[cpu_failed], failedcpus, sr<#2>(<#1>)
  W cpubreak_idle<#1>;
<1..NS> IF sr1==# FROM sr1(#), cpu#[cpu_failed], failedcpus
           TO cpu#[cpu_idle], freecpus W cpurepair#;
<2..NS> IF sr<#-1>==0 FROM sr#(sr#) TO sr<#-1>(sr#);
// ========================================== Results ====
<1..NS> RESULT>> if(cpu#==cpu_busy) cpuutil# += PROB;
<1..NS> RESULT>> if(cpu#==cpu_busy) busycpus += PROB;
<1..NS> RESULT>> if(cpu#==cpu_idle OR cpu#==cpu_busy) goodcpus+=PROB;
<1..NS> RESULT>> if(cpu#==cpu_failed) nfailedcpus += PROB;
RESULT if(busy_terminals>0) busyterm += PROB*busy_terminals);
RESULT>> termutil = busyterm / NT;
RESULT>> if(retrying_terminals>0) retravg += (PROB*retrying_terminals);
RESULT>> if(failedcpus>0) repairutil += PROB;
RESULT if(waiting_terminals>0) waitall += (PROB*waiting_terminals);
RESULT>> resptime = (retravg + waitall) / NT / (prgen * termutil);
RESULT>> overallutil = busycpus + termutil*NT + repairutil;
```

B. The MOSEL Model for the Fastest Free Server Policy

```
// ============================== Constant definitions ====
#define NT 20
#define NS 4
// ====================== Variables (input parameters) ====
VAR double prgen;
VAR double prretr;
<1..NS> VAR double prrun#;
<1..NS> VAR double cpubreak_idle#;
<1..NS> VAR double cpubreak_busy#;
<1..NS> VAR double cpurepair#;
// ================================== Node definitions ====
enum cpu_states {cpu_busy, cpu_idle, cpu_failed};
NODE busy_terminals[NT] = NT;
NODE retrying_terminals[NT] = 0;
NODE waiting_terminals[NS] = 0;
<1..NS> NODE cpu#[cpu_states] = cpu_idle;
        NODE freecpus[NS] = NS;
        NODE failedcpus[NS] = 0;
<1..NS> NODE sr#[NS] = 0;
// ======================================= Transitions ====
FROM cpu1[cpu_idle], busy_terminals, freecpus
        TO cpu1[cpu_busy], waiting_terminals
        W prgen*busy_terminals;
FROM cpu2[cpu_idle], busy_terminals, freecpus
        TO cpu2[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy W prgen*busy_terminals;
FROM cpu3[cpu_idle], busy_terminals, freecpus
        TO cpu3[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy AND cpu2==cpu_busy
        W prgen*busy_terminals;
FROM cpu4[cpu_idle], busy_terminals, freecpus
        TO cpu4[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy AND cpu2==cpu_busy AND cpu3==cpu_busy
        W prgen*busy_terminals;
FROM busy_terminals TO retrying_terminals
        IF freecpus==0 W prgen*busy_terminals;
FROM cpu1[cpu_idle], retrying_terminals, freecpus
        TO cpu1[cpu_busy], waiting_terminals
        W prretr*retrying_terminals;
FROM cpu2[cpu_idle], retrying_terminals, freecpus
        TO cpu2[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy W prretr*retrying_terminals;
FROM cpu3[cpu_idle], retrying_terminals, freecpus
        TO cpu3[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy AND cpu2==cpu_busy
        W prretr*retrying_terminals;
FROM cpu4[cpu_idle], retrying_terminals, freecpus
        TO cpu4[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy AND cpu2==cpu_busy AND cpu3==cpu_busy
        W prretr*retrying_terminals;
<1..NS><NS> FROM cpu<#1>[cpu_busy], waiting_terminals{
  TO cpu<#1>[cpu_idle], busy_terminals, freecpus W prrun<#1>;
  TO cpu<#1>[cpu_failed],retrying_terminals,failedcpus, sr<#2>(<#1>)
  W cpubreak_busy<#1>; }
<1..NS><NS> FROM cpu<#1>[cpu_idle], freecpus
  TO cpu<#1>[cpu_failed], failedcpus, sr<#2>(<#1>) W cpubreak_idle<#1>;
<1..NS> IF sr1==# FROM sr1(#), cpu#[cpu_failed], failedcpus
           TO cpu#[cpu_idle], freecpus W cpurepair#;
<2..NS> IF sr<#-1>==0 FROM sr#(sr#) TO sr<#-1>(sr#);
// ========================================== Results ====
<1..NS> RESULT>> if(cpu#==cpu_busy) cpuutil# += PROB;
<1..NS> RESULT>> if(cpu#==cpu_busy) busycpus += PROB;
<1..NS> RESULT>> if(cpu#==cpu_idle OR cpu#==cpu_busy) goodcpus+=PROB;
<1..NS> RESULT>> if(cpu#==cpu_failed) nfailedcpus += PROB;
RESULT if(busy_terminals>0) busyterm += PROB*busy_terminals);
RESULT>> termutil = busyterm / NT;
RESULT>> if(retrying_terminals>0) retravg += (PROB*retrying_terminals);
RESULT>> if(failedcpus>0) repairutil += PROB;
RESULT if(waiting_terminals>0) waitall += (PROB*waiting_terminals);
RESULT>> resptime = (retravg + waitall) / NT / (prgen * termutil);
RESULT>> overallutil = busycpus + termutil*NT + repairutil;
```