Annales Univ. Sci. Budapest., Sect. Comp. 23 (2004) 213-227

HOMOGENEOUS FINITE–SOURCE RETRIAL QUEUES WITH SERVER AND SOURCES SUBJECT TO BREAKDOWNS AND REPAIRS

J. Roszik (Debrecen, Hungary)

Abstract. The aim of this paper is to investigate a retrial queueing system with a finite number of homogeneous non-reliable sources of calls and a single non-reliable server, that is the sources and the server are subject to random breakdowns and repairs. The novelty of the present paper is the non-reliability of the sources.

The main performance and reliability measures are derived, and the MOSEL tool is used to formulate the model and to calculate these measures. Several numerical results are graphically displayed to illustrate the effect of the non-reliability of the sources and the server on some of the system measures.

All random variables involved in the model construction are assumed to be exponentially distributed and independent of each other.

1. Introduction

Retrial queues (or queueing systems with repeated attempts) are characterized by the following feature: a primary request finding all servers busy upon arrival leaves the service area but after some random time repeats his demand. This feature plays a special role in many computer and communication systems and networks as well as in other practical applications. For more detailed information and results on this type queueing systems, see for example [7], [10], [11]. We mention that there are possible other rules for the repetition of

Research is partially supported by German-Hungarian Bilateral Intergovernmental Scientific Cooperation, OMFB-DLR No. 21-2000, Hungarian Scientific Research Found OTKA T0-34280/2000 and FKFP grant 0191/2001.

demand, e.g. [18], [19] consider the case of constant repetition time (so-called cyclic-waiting systems), a comparison of classical retrial and cyclic-waiting systems is given in [20].

Finite-source queueing models with quasi-random input are often used for performance evaluation of computer and communication systems. A complete survey can be found on queueing systems without retrials in [22]. The first paper concerning finite-source retrial queues was published by Kornyshev [15] in 1969, then there was a rapid growth in a number of papers dealing with them. For some fundamental results on this topic see [6], [11], [12], [13], [14].

The components of the real systems may be subject to random breakdowns (see [16], [23]), so it is important to investigate non-reliable queueing systems (see [1], [3], [21]) as well as non-reliable retrial queues where the sources and the server may be subject to random breakdowns and repairs. Non-reliable infinite-source retrial queues were studied, for example in the works [5], [8], [17], [24] and finite-source retrial queues in [4].

The purpose of this paper is to give the main stationary performance and reliability measures of the non-reliable model described in the next section, and to illustrate graphically the effect of changing various parameters on them. The performance and reliability modeling tool MOSEL (Modeling, Specification and Evaluation Language), see [9], is used to formulate the model and to obtain the system performance measures. The analytical results can be graphically displayed using IGL (Intermediate Graphical Language) which belongs to the MOSEL tool.

This paper can be regarded as the continuation of the work [4] that analyzes the homogeneous finite-source retrial queues with reliable sources and a single server subject to breakdowns and repairs.

The organization of the paper is as follows. Section 2 contains an accurate description of the model, the derived main performance and reliability measures, the MOSEL implementation of the model and its short explanation and validation. Section 3 is devoted to some graphically displayed numerical results with comments. Finally, the paper ends with a conclusion.

2. The $M/M/1//K$ retrial queueing model with non-reliable server and sources

Consider a finite-source single server retrial queueing system, where primary calls are generated by K $(1 < K < \infty)$ sources. The server can be in

operational (up) or non-operational (down) states, and it can be idle and busy. Each of the sources can be in four states: generating a primary call (busy), sending repeated calls, under service and failed. If a source is busy at time t, it can generate a primary call during interval $(t, t+dt)$ with probability $\lambda dt + o(t)$. If the server is up and idle at the time of the arrival of a call then the call starts to be served immediately, the source moves into the under service state and the server moves into busy state. The service is finished during the interval $(t, t+dt)$ with probability $\mu dt + o(t)$ if the server is available.

The server can fail during the interval $(t, t+dt)$ with probability $\delta dt + o(t)$ if it is idle, and with probability $\gamma dt + o(t)$ if it is busy. If $\delta = 0,\ \gamma > 0$ or $\delta = \gamma > 0$ *active or independent breakdowns* can be discussed, respectively. If the server fails in busy state, it either *continues servicing* the interrupted call after it has been repaired or the interrupted request *returns to the orbit*. The repair time of the server is exponentially distributed with a finite mean $1/\tau$. If the server is failed two different cases can be treated. Namely, *blocked sources* case when all the operations are stopped except from the repair of the server. In the *non-blocked (intelligent) sources* case only service is interrupted, but all the other operations are continued.

If the server is busy (or failed in the non-blocked case) at the time of the arrival of a call then the source starts generation of a Poisson flow of repeated calls with rate ν until it finds the server free and up. After service the source becomes busy and it can generate a new primary call, and the server becomes idle, so it can serve a new call.

Sources can be non-operational only in busy state. If a source is busy at time t it can fail during the interval $(t, t+dt)$ with probability $\eta dt + o(t)$ and then it moves to the repairman who follows FIFO discipline for the source breakdowns and gives preemptive priority to the server failure. The repair time of the sources is exponentially distributed with a finite mean $1/\kappa$. All the times involved in the model are assumed to be mutually independent of each other.

2.1. The underlying Markov chain

The system state at time t can be described with the process $X(t) = \{Y(t);\ C(t);\ N(t);\ Z(t)\}$, where $Y(t) = 0$ if the server is up, $Y(t) = 1$ if the server is failed, $C(t) = 0$ if the server is idle, $C(t) = 1$ if the server is busy, $N(t)$ is the number of sources of repeated calls and $Z(t)$ is the number of failed sources at time t. Because of the exponentiality of the involved random variables 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 values of the rate of generation of primary calls. From now on we will assume that the system is in the steady state.

We define the stationary probabilities:

$$P(q;r;j;k) = \lim_{t\to\infty} P\{Y(t)=q,\ C(t)=r,\ N(t)=j,\ Z(t)=k\},$$

$$q=0,1 \quad r=0,1, \quad j=0,\ldots,K^*, \quad k=0,\ldots,K-r-j,$$

where

$$K^* = \begin{cases} K-1 & \text{for blocked case,} \\ \\ K-r & \text{for non-blocked case.} \end{cases}$$

To obtain the performance and reliability measures the tool MOSEL is used to get the state probabilities in the equilibrium. In this way, by implementing the model in MOSEL, we get these probabilities more easily and make the model more practically usable.

Once we have obtained the steady state probabilities the main system performance measures can be derived in the following way:

- *The availability of the server*

$$A_S = \sum_{r=0}^{1}\sum_{j=0}^{K^*}\sum_{k=0}^{K-r-j} P(0,r,j,k).$$

- *The mean number of sources of repeated calls*

$$N = E[N(t)] = \sum_{q=0}^{1}\sum_{r=0}^{1}\sum_{j=0}^{K^*}\sum_{k=0}^{K-r-j} jP(q,r,j,k).$$

- *The mean number of calls staying in the orbit or in service*

$$M = E[N(t)+C(t)] = N + \sum_{q=0}^{1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1-j} P(q,1,j,k).$$

- *The mean number of operational sources*

$$N_O = K - \sum_{q=0}^{1}\sum_{r=0}^{1}\sum_{j=0}^{K^*}\sum_{k=1}^{K-r-j} kP(q,r,j,k).$$

- *The utilization of the server*

$$U_S = \sum_{j=0}^{K-1}\sum_{k=0}^{K-1-j} P(0,1,j,k).$$

- *The utilization of the repairman*

$$U_R = \sum_{r=0}^{1} \sum_{j=0}^{K^*} \sum_{k=1}^{K-r-j} P(0,r,j,k) + \sum_{r=0}^{1} \sum_{j=0}^{K^*} \sum_{k=0}^{K-r-j} P(1,r,j,k).$$

- *The utilization of the sources*

$$U_{SO} = \begin{cases} \dfrac{N_O - M}{K} A_S & \text{for blocked case,} \\ \dfrac{N_O - M}{K} & \text{for non-blocked case.} \end{cases}$$

- *The overall utilization*

$$U_O = U_S + KU_{SO} + U_R.$$

- *The mean rate of generation of primary calls*

$$\overline{\lambda} = \begin{cases} \lambda E[K - C(t) - N(t) - Z(t);\ Y(t) = 0] & \text{for blocked case,} \\ \lambda E[K - C(t) - N(t) - Z(t)] & \text{for non-blocked case.} \end{cases}$$

- *The mean response time*

$$E[T] = M/\overline{\lambda}.$$

- *The mean waiting time*

$$E[W] = N/\overline{\lambda}.$$

- *The blocking probability of a primary call*

$$B = \begin{cases} \dfrac{\lambda E[K - C(t) - N(t) - Z(t);\ Y(t) = 0;\ C(t) = 1]}{\overline{\lambda}} & \text{for blocked case,} \\ \dfrac{\lambda E[K - C(t) - N(t) - Z(t);\ C(t) = 1]}{\overline{\lambda}} & \text{for non-blocked case.} \end{cases}$$

2.2. The MOSEL implementation

This section demonstrates how this queueing system can be modeled and the main performance measures can be calculated by the MOSEL (Modeling, Specification and Evaluation Language) tool. The technical details of

programming can be seen in [9]. The following MOSEL program belongs to the type of model where the service is continued after the server's breakdown and repair, and operations are blocked during the server is not operational. It does not contain the picture section, which is needed to generate various graphical representations of the measures. The figures in the next section were automatically generated by the tool with the corresponding picture part. The following terminology is used in the MOSEL program: the server and the sources are referred as a CPU and terminals.

```
/* retrialnr-hom-cont.msl begins */
/*-----------------------Definitions-----------------------*/
#define NT 5

/*=============== No changes required below ==================*/
VAR double prgen;
VAR double prretr;
VAR double prrun;
VAR double cpubreak_idle;
VAR double cpubreak_busy;
VAR double cpurepair;
VAR double termbreak;
VAR double termrepair;
enum cpu_states {cpu_busy, cpu_idle};
enum cpu_updown {cpu_up, cpu_down};

/*---------------- Node definitions ------------------------*/
NODE busy_terminals[NT] = NT;
NODE retrying_terminals[NT] = 0;
NODE waiting_terminals[1] = 0;
NODE failed_terminals[NT] = 0;
NODE cpu_state[cpu_states] = cpu_idle;
NODE cpu[cpu_updown] = cpu_up;

/*----------------- Transitions ----------------------------*/
IF cpu==cpu_up FROM cpu_idle, busy_terminals{
    TO cpu_busy, waiting_terminals W prgen*busy_terminals;
    TOM cpu_idle, failed_terminals W termbreak*busy_terminals;
}
IF cpu==cpu_up AND cpu_state==cpu_busy FROM busy_terminals{
```

```
    TO retrying_terminals W prgen*busy_terminals;
    TO failed_terminals W termbreak*busy_terminals;
}
IF cpu==cpu_up FROM cpu_idle, retrying_terminals
    TO cpu_busy, waiting_terminals W prretr*retrying_terminals;
IF cpu==cpu_up FROM cpu_busy, waiting_terminals
    TO cpu_idle, busy_terminals W prrun;
IF cpu_state==cpu_idle FROM cpu_up TO cpu_down W cpubreak_idle;
IF cpu_state==cpu_busy FROM cpu_up TO cpu_down W cpubreak_busy;
FROM cpu_down TO cpu_up W cpurepair;
IF cpu==cpu_up FROM failed_terminals
    TO busy_terminals W termrepair;

/*------------------------- Results ------------------------*/
RESULT>> if (cpu==cpu_up AND cpu_state==cpu_busy) cpuutil += PROB;
RESULT>> if (cpu==cpu_up) goodcpu += PROB;
RESULT>> if (cpu==cpu_up) busyterm += (PROB*busy_terminals);
RESULT>> termutil = busyterm / NT;
RESULT>> if (cpu==cpu_up) retravg += (PROB*retrying_terminals);
RESULT>> if (failed_terminals>0)
            failedtermavg += (PROB*failed_terminals);
RESULT>> goodterminals = NT - failedtermavg;
RESULT>> if (cpu==cpu_down OR failed_terminals>0)
            repairutil += PROB;
RESULT>> if (waiting_terminals>0) waitall +=
                (PROB*waiting_terminals);
RESULT>> if (retrying_terminals>0)
            retrall += (PROB*retrying_terminals);
RESULT>> resptime =
            (retrall + waitall) / NT / (prgen * termutil);
RESULT>> overallutil = cpuutil + busyterm + repairutil;
/* retrialnr-hom-cont.msl ended */
```

The **declaration part** defines the number of terminals *(NT)*, this is the only program code line, that must be modified when modeling larger systems. We define the five parameters for the terminals: *prgen* denotes the rate of primary call generation, *prretr* references to the rate of repeated call generation, *prrun* denotes the service rate, and *termbreak* and *termrepair* denote the failure

and repair rates. The *cpubreak_idle, cpubreak_busy* and *cpurepair* variables denote the failure rate in the two CPU states and the repair rate for the CPU. The value of the terminal and CPU parameters must be given when the program is running. The CPU has two states: idle and busy, and it can be up or failed in both states.

The **node part** defines the nodes of the system. Our queueing network contains 6 nodes: one node for the number of busy terminals (primary call generation, every terminal is busy when the system starts), the number of retrying terminals (repeated call generation), the number of waiting terminals (job service at the CPU) and the number of failed terminals, respectively, and two pieces for the CPU (which is idle and up at the starting time).

The **transition part** describes how the system works. The first transition rule defines the successful primary call generation: the CPU moves from the idle state to busy and the terminal from busy to waiting. During primary call generation the terminal can fail. In this case the CPU remains idle, and the terminal moves to state failed. The second rule shows an unsuccessful primary call generation: if the CPU is busy when the call is generated then the terminal moves to state retrying. The terminal can fail like in the first rule. The third rule handles the case of the successful repeated call generation: the CPU moves from the idle state to busy and the terminal from retrying to waiting. The fourth rule describes the request service at the CPU. The fifth and sixth rules describe the CPU fail in idle and busy state and the seventh rule shows the CPU repair. The last rule shows the terminal repair: the terminal moves to state busy, so it can generate a new primary call. Terminals can be repaired only if the CPU is operational.

Finally the **result part** calculates some output performance measures.

2.3. Validation of results

The results in the reliable case were validated by the Pascal program given in [11]. In the case of server's breakdowns and reliable sources the program was tested by the results of [4].

In Table 1 some test results are collected when the retrial rates are quite large. The corresponding performance measures should be very close to each other in the case of continued service, restarted repeated call generation after server failure (abbreviated by orbit) and the FIFO discipline which was studied in [2]. As we can see, the results confirm our expectation, the derived results are the same up to the 6th decimal digit.

	non–rel. retrial(cont.)	non–rel. retrial(orbit)	non–rel. FIFO
Number of sources:	3	3	3
Request's generation rate:	0.1	0.1	0.1
Service rate:	1	1	1
Retrial rate:	1e+25	1e+25	–
Server's failure rate:	0.02	0.02	0.02
Server's repair rate:	0.05	0.05	0.05
Sources' failure rate:	0.03	0.03	0.03
Sources' repair rate:	0.05	0.05	0.05
Utilization of the server:	0.0965679029	0.0965679117	0.0965678743
Mean response time:	1.5546014407	1.5546014565	1.5546013953

Table 1. Validations

3. Numerical examples

In this section we consider some sample numerical results to illustrate graphically the influence of the non-reliable server and sources on the mean response time $E[T]$, the overall system utilization, the mean number of sources of repeated calls and the mean number of operational sources. The input parameters of the following figures are collected in Table 2.

	NT	λ	μ	ν	δ	γ	τ	η	κ
Figure 1	5	0.8	4.5	0.5	0.05	x axis	0.1	0.06	0.15
Figure 2	5	0.1	0.5	x axis	0.05	0.05	0.1	0.06	0.15
Figure 3	5	0.8	4.5	x 0.5	0.05	0.05	0.1	x axis	0.15

Table 2. Input parameters

3.1. Comments

- In Figure 1 the effect of the server's failure is displayed in the continuous cases. The almost linear increase in $E[T]$ in each case is because the response time is the sum of the down time of the server, the service and repeated call generation time of the request (which do not change during the failure), thus the failure has a linear effect on this measure.

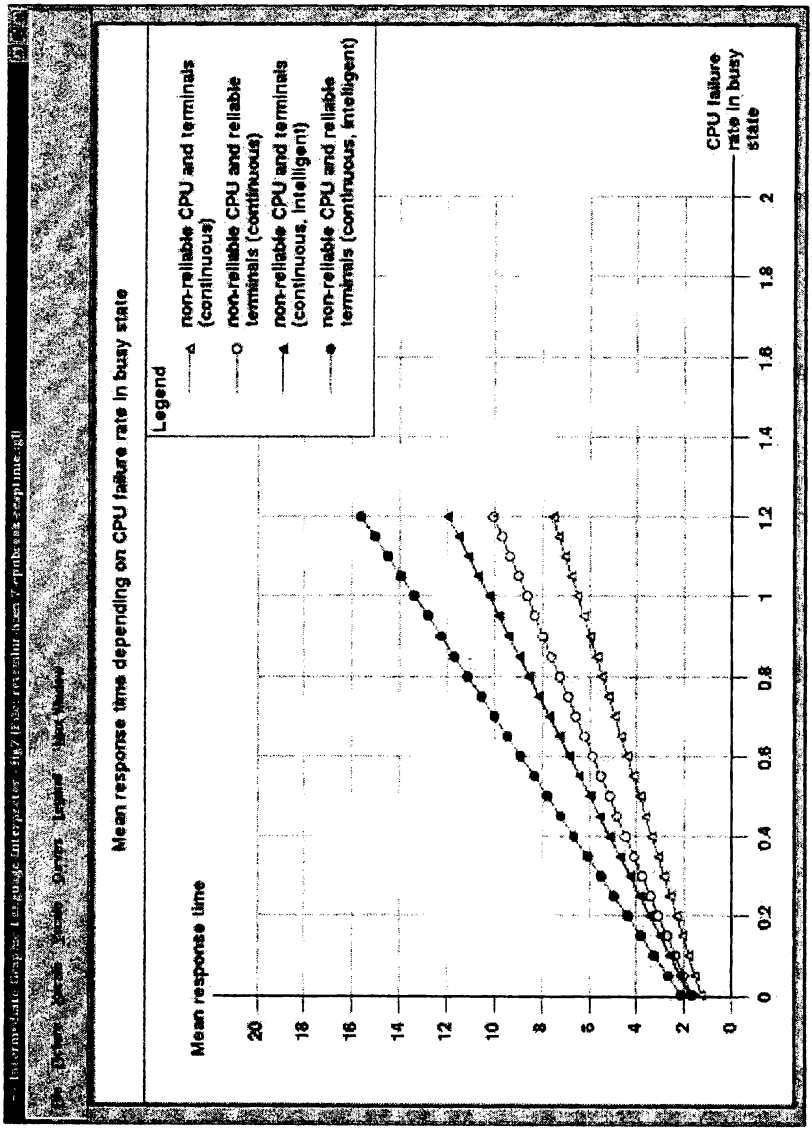

Figure 1. $E[T]$ versus CPU failure rate in busy state

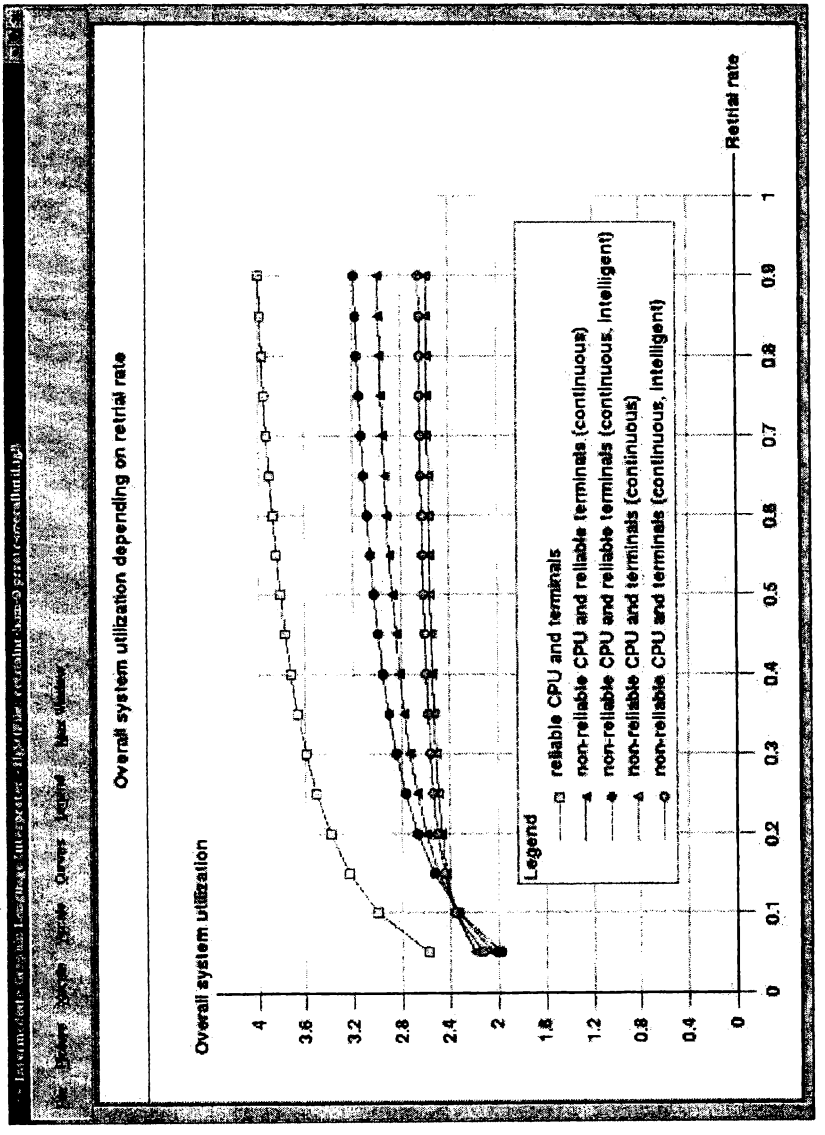

Figure 2. Overall utilization versus retrial rate

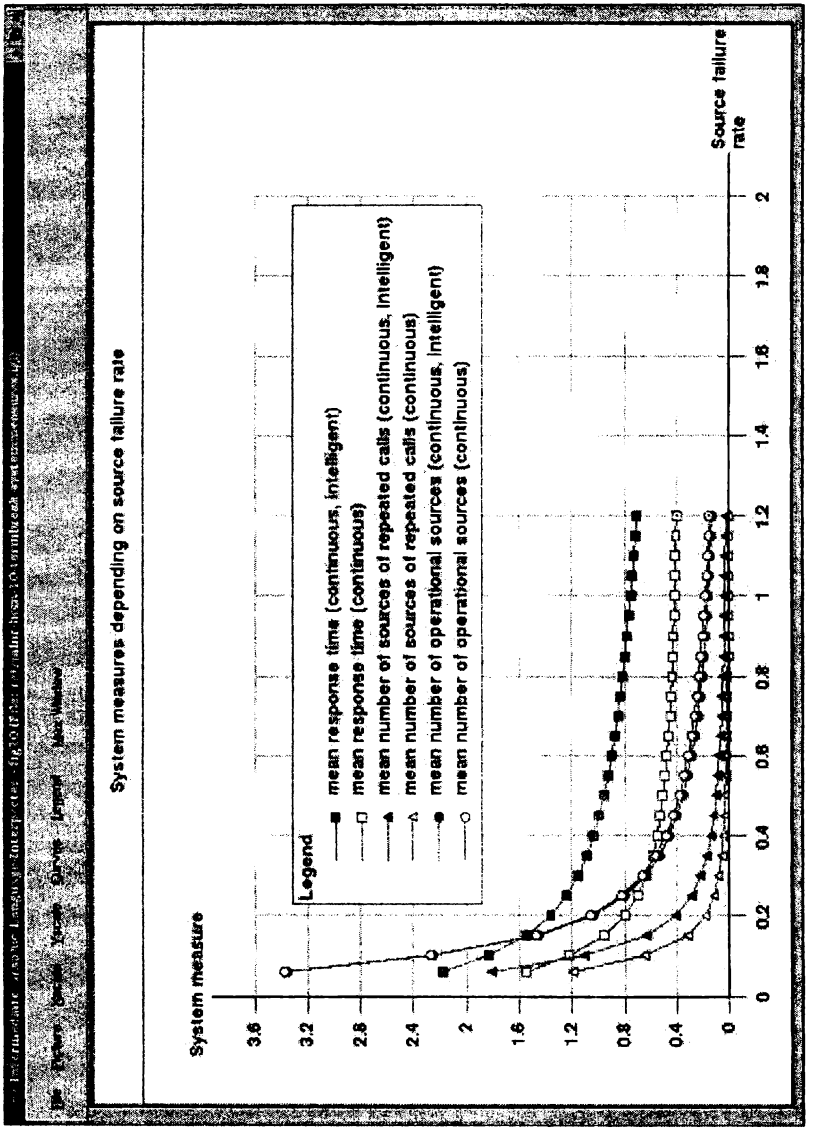

Figure 3. System measures versus source failure rate

- In Figure 2 we can see the effect of the retrial rate on the overall system utilization. At the beginning the overall system utilizations are larger for the continuous than the non-continuous cases, then it changes and the difference increases as the retrial rate increases.
- In Figure 3 the effect of the sources' failure rate is displayed on the mean response time, the mean number of sources of repeated calls and the mean number of operational sources. There is a very slight difference between the continuous and non-continuous cases for the mean number of operational sources, and the mean response time decreases as the failure rate increases, that is the the mean number of operational sources decreases.

4. Conclusion

In this paper a homogeneous finite-source retrial queueing system with non-reliable sources and a single non-reliable server is studied. The novelty of the investigation is the non-reliability of the sources which makes the system rather complicated. The MOSEL tool was used to formulate the model and to calculate some system measures which were graphically displayed to show the effect of the non-reliability of the server and the sources on the mean response times of the calls, the overall system utilization, the mean number of sources of repeated calls and the mean number of operational sources.

Acknowledgement. Special thanks to Béla Almási and János Sztrik, their help is greatly acknowledged.

References

[1] **Almási B.,** A queueing model for a processor-shared multi-terminal system subject to breakdowns, *Acta Cybernetica,* **10** (4) (1993), 273-282.

[2] **Almási B., Bolch G. and Sztrik J.,** Performability modeling of non-homogeneous terminal systems using MOSEL, *5th International Workshop of Performability Modeling of Computer and Communication Systems, Erlangen, Germany, 2001,* 37-41.

[3] **Almási B. and Sztrik J.,** Optimization problems on the performance of a non-reliable terminal system, *Computers and Mathematics with Applications,* **38** (1999), 13-21.

[4] **Almási B., Roszik J. and Sztrik J.,** *Homogeneous finite-source retrial queues with server subject to breakdowns and repairs,* Technical report, University of Debrecen, 2002/17.

[5] **Artalejo J.R.,** New results in retrial queueing systems with breakdown of the servers, *Statistica Neerlandica,* **48** (1994), 23-36.

[6] **Artalejo J.R.,** Retrial queues with a finite number of sources, *J. Korean Math. Soc.,* **35** (1998), 503-525.

[7] **Artalejo J.R.,** Accessible bibliography on retrial queues, *Mathematical and Computer Modelling,* **30** (1999), 1-6.

[8] **Aissani A. and Artalejo J.R.,** On the single server retrial queue subject to breakdowns, *Queueing Systems,* **30** (1998), 309-321.

[9] **Begain K., Bolch G. and Herold H.,** *Practical performance modeling, application of the MOSEL language,* Kluwer, 2001.

[10] **Falin G.I.,** A survey of retrial queues, *Queueing Systems,* **7** (1990), 127-168.

[11] **Falin G.I. and Templeton J.G.C.,** *Retrial queues,* Chapman and Hall, London, 1997.

[12] **Falin G.I. and Artalejo J.R.,** A finite source retrial queue, *European J. of Operational Research,* **108** (1998), 409-424.

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

[14] **Gomez Corral A.,** Analysis of a single-server retrial queue with quasi-random input and nonpreemptive priority, *Computers and Mathematics with Applications,* **43** (2002), 767-782.

[15] **Kornyshev Y.N.,** Design of a fully accessible switching system with repeated calls, *Telecommunications,* **23** (1969), 46-52.

[16] **Kovalenko I.N., Kuznetsov N.Yu. and Pegg P.A.,** *Mathematical theory of reliability of time dependent systems with practical applications,* John Wiley and Sons, Chichester, 1997.

[17] **Kulkarni V.G. and Choi Bong Dae,** Retrial queues with server subject to breakdowns and repairs, *Queueing Systems Theory and Applications,* **7** (1990), 191-208.

[18] **Lakatos L.,** On a simple continuous cyclic-waiting problem, *Annales Univ. Sci. Budapest. Sect. Comp.,* **14** (1994), 105-113.

[19] **Lakatos L.,** A retrial system with time-limited tasks, *Theory of Stochastic Processes,* **8(24)** (3-4) (2002), 249-255.

[20] **Mykhalevych K.V.,** A comparison of a classical retrial $M/G/1$ queueing system and a Lakatos-type $M/G/1$ cyclic-waiting time queueing system, *Annales Univ. Sci. Budapest. Sect. Comp.,* **23** (2003), 229-238.

[21] **Sztrik J. and Gál T.,** A recursive solution of a queueing model for a multi-terminal system subject to breakdowns, *Performance Evaluation,* **11** (1990), 1-7.

[22] **Takagi H.,** *Queueing analysis. A foundation of performance evaluation, Vol. 2. Finite systems,* North-Holland, 1993.

[23] **Trivedi K.S.,** *Probability and statistics with reliability, queueing and computer science applications,* Prentice Hall, Englewood Cliffs, 1982.

[24] **Wang Jinting, Cao Jinhua and Li Quanlin,** Reliability analysis of the retrial queue with server breakdowns and repairs, *Queueing Systems Theory and Applications,* **38** (2001), 363-380.

(Received April 28, 2003)

J. Roszik
Institute of Informatics
University of Debrecen
H-4010 Debrecen, P.O.B. 12
Hungary
jroszik@delfin.unideb.hu