SEU effect analysis in a open-source router via a distributed fault injection environment

SEU effect analysis in a open-source router via a distributed fault injection environment

Authors: Benso A., Di Carlo S., Di Natale G., Prinetto P.,

Published in the Proceedings of the Design, Automation and Test in Europe, Conference and Exhibition (DATE), 13-16 Mar. 2001, Munich, DE.

DOI: 10.1109/DATE.2001.915028

SEU Effect Analysis in a Open-Source Router via a Distributed Fault Injection Environment

Alfredo BENSO, Stefano DI CARLO, Giorgio DI NATALE, Paolo PRINETTO

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24 - I-10129, Torino, Italy
Email: { benso, dicarlo, dinatale, prinetto }@polito.it
http://www.testgroup.polito.it

Abstract

The paper presents a detailed error analysis and classification of the behavior of an open-source router, when affected by Single Event Upsets (SEUs). The experimental results have been gathered on a real communication network, resorting to an ad-hoc Fault Injection system. The injector has been designed to corrupt the router during its normal service and to analyze the SEU injection effects on the overall distributed system.

The performed experiments allowed the authors to identify the most critical memory regions and to cluster the router variables according to their impact on system dependability.

1. Introduction

The increased use of computer systems in applications that require very high dependability is now widespread and commonly accepted.

We are nowadays entering a world of global communications, in which a multi-faceted variety of services, ranging from the simple voice transmission to the real-time exchange of enormous volumes of data, is offered to a potentially huge number of customers.

Examples of these services include video-conferencing, Web browsing, VoIP, but there is a general feeling that the new communication systems will trigger the expansion of many kinds of services that are today completely unforeseen.

One of the crucial points of this revolution is represented by the deployment of *wide-band reliable communication infrastructures*. This appears to be as one of the major challenges for the communication society. The deployment of global communication systems that ensure adequate levels of performance and dependability in data communications is a *conditio sine qua non* for the opening of the new scenarios of the future service society.

The quality of services accessed through a communication network may be impaired by the malfunctioning that arises from failures of the hardware/software network infrastructure components, as well as from the erroneous behavior/interaction of the applications that use the network to transfer the data.

Since from an economical point of view hardware redundancy is not always the most practical solution, the use of dependable software techniques is an effective and low-cost solution to develop high dependable communication systems. In order to target the best dependable software techniques, an analysis of the behavior of the communication system when faults occur is needed.

Some of the most critical components inside a communication network are the routing devices. In fact, a failure in one of this components can affect an entire subset of the network making impossible the communication between two points especially inside network not strongly connected.

The objective of this work is the study and the classification of the behavior of an *Open Source Router* when a *SEU (Single Error Upset)* affects it. The proposed work aims at contributing to the definition of methodologies for enhancing the quality of services that are provided through network communication systems.

The realization of the study proposed in this paper relies on a Fault Injection environment to inject faults inside the memory of the router and to observe its behavior in presence of a fault. Due to the distributed nature of the target system, the Fault Injector must be distributed as well, to collect results not only in the router but also in the clients connected to the router.

The interest in Fault Injection techniques has lead to many researches about its applicability, validity, and possible applications. Many Fault Injection tools have been developed in the context of different researches. *Messaline* [1] has a design based on a formalized Fault Injection methodology. The result is a flexible testbed

capable of simultaneously injecting multiple, pin-level faults into different target systems to collect coverage, latency, and error-propagation measurements. *Fiat* [2] and *Ferrari* [3] use software-implemented injections to emulate hardware faults. The *Focus* simulation environment [4] conducts fault sensitivity experiments on chip-level designs. Transient faults are injected through a runtime modification of the circuit. *Depend* [5] is a process-based simulator that provides a library of objects to behaviorally model a system's hardware component. *React* [6] is a software testbed that abstracts multiprocessor systems at the architectural level.

Despite the effectiveness of the previously listed tools, they are designed especially to inject faults into hardware systems. Our study requires a fault injection tool able to target distributed software systems.

The paper is organized as follows: Section 2 defines the network architecture, whereas Section 3 summarizes the proposed fault model. Section 4 describes the implementation of the fault injector. Sections 5, 6 present the experimental results, whereas Section 7 exploits some conclusion.

2. The test case

To set up a significant experiment, the following problems have to be faced:

- selection of a "significant" network architecture,
- selection of a proper router implementation,
- selection of suitable traffic on the network in terms of both protocols and data.

2.1. Network architecture

One of the key points of the study proposed in this paper is the definition of the architecture of the network used to gather experimental results. Two opposite constraints need to be fulfilled. On one hand, the network should be simple enough to be easily manageable and controllable. On the other hand, it should be complex enough to reflect the characteristics of a real network. Trading-off these requirements, we selected the network architecture shown in Figure 1.

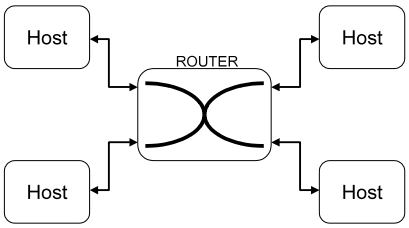

Figure 1: Network Architecture

Despite its simplicity, the adopted architecture can reflect a real situation. In fact, if each host is considered not just as a single node of the network but as a complex subnet, we obtain a real and common scenario in which a certain number of different networks are connected together via a router, as shown in Figure 2.

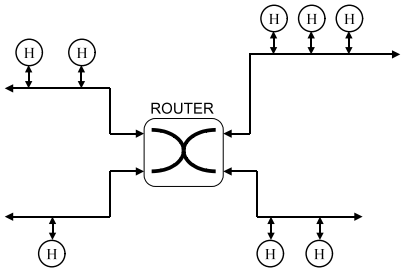

Figure 2: Expanded Network Architecture

Each host then can be fatherly expanded, modeling the global networks mentioned in Section 1.

2.2. Router implementation

Looking at the proposed architecture, the key role of the router easily comes up. A fault in the router can totally isolate some portions of the network.

Concerning the router implementation, the use of commercial devices is not feasible due to the usage of proprietary software, which makes necessary the use of expensive and complex hardware Fault Injector.

Thus we opted for an *Open-Source* Router implemented on a PC running the Linux Operating System [7], equipped with multiple network interfaces. The advantage of using a Linux system is the possibility to access directly the router code and to use software fault injection techniques instead of the hardware ones.

2.3. Traffic emulation

To be able to deal with worst cases, the UDP (User Datagram Protocol) [8] will be used. UDP is a non-reliable transport protocol, thus not capable of correcting routing errors occurring in the transmitted packets.

As far as the transmitted data are concerned, each client sends and receives packets to and from all the others clients. As previously explained, the datagrams are sent using the UDP protocol. The format of the payload is shown in Figure 3.

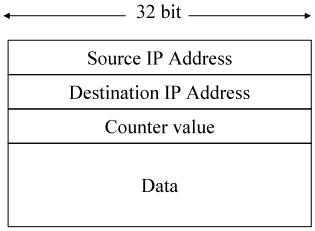

Figure 3: Datagram Payload Format

The first two fields contain the source and the destination address, respectively. The third field is a counter: its starting value is selected randomly the first time a datagram is sent, and then incremented each time a new datagram is sent to the same client. In this way, each client can check whether it received all the packets addressed to it, or not. The last field contains a predefined string, whose variable length is calculated as a function of the values of the previous fields.

3. Fault Model

The adopted fault model for the fault injection experiments is the transient fault *Single Error Upset* (SEU), consisting in temporally flipping one bit in one data memory location.

The question of how much this fault model represents real pathologies induced by the occurrence of real defects is crucial. Several software-implemented fault injection studies are dedicated to the analysis of the relationship between fault injected by software and physical faults. In particular, both NASA [9] and IBM [10] made statistical studies about the most common error occurring in modern digital circuits. These studies lead to the conclusion that, due to the high miniaturization and the high work frequencies, today circuits are becoming more and more susceptible to the effect of ionizing radiation and noise source. Moreover, [9] and [10] reported SEU be one of the main observed effects.

The effectiveness of the used fault model is increased when dealing with space applications, where routers need to be installed on satellites, where the probability of SEU is very high.

4. Fault Injector

To analyze the behavior of the network when a failure occurs, a system able to insert fault inside the router memory is needed.

A usual problem in setting up Fault Injection environments is the definition of a significant *fault injection policy*, in terms of *where* and *when* faults have to be injected. Analyzing the router memory structure we identified four different memory areas:

1. *Network Packet*
2. *Code Segment*
3. *Routing Table*
4. *Local Data Segment*.

Before entering implementation details, let's make a brief analysis of the potential effects of faults in these different areas.

Errors detection in the network packets is a well know problem in telecommunication community. All the standard protocols used in the communication network implement error detection and correction mechanisms. Thus, an error analysis in this area is not necessary.

All the others three areas are instead candidate targets for the fault injection experiments. In particular, the Code Segment and Local Data Segment need to be well analyzed.

As a consequence of the injection of a fault, one can expect the following three main behaviors of the router:

- *The router still works properly:* in this case the fault is either still latent or it has been overridden. These faults are not interesting because they don't affect the router behavior.
- *The router still works, but the packets are not routed correctly:* these errors are most likely due to a fault in the router's Routing Table. Once again, this kind of faults is not so critical since, due to their occurrence, packets are either routed on longer paths or, in the worst case, lost. To investigate anyhow this class of faults, as described in section 2.3, we adopted the UDP Protocol that, being a non-reliable protocol, is not able to correct routing errors.
- *The router crashes:* this is the worst situation due to the isolation of some network areas and to the consequent high recovery time.

To fulfill the above-mentioned constraints, the Fault Injector has been implemented as a kernel daemon of the Linux OS. The fault injection process is split into several steps, summarized by the pseudo-code in Figure 4.

```
/*Identifythe memory portion (active memory)
where the faults have to be injected.*/
Select_memory();
For each byte in active_memory
 {
/*Select A bit in a random position.*/
  Select_bit();
/*Save the content of the environment
variables.*/
  Save_status();
  /* Flip the selected bit (SEU injection)*/
  Bit_flip();
/* The fault effects are looked for a time
window of 5 seconds.*/
  Wait for 5 sec
/*If a crash does not occur, restore the
saved safe state. */
  Reload_status();
 }
```

Figure 4: Fault Injector Pseudo Code

During the fault injection all the clients connected transmit and receive a lot of datagrams, to verify the effects of the fault on the network behavior.

When a client receives a datagram, verifying its correctness, it can detect:

- errors due to packets loss, using the datagram counter
- transmission error, checking the string length.

System crashes can be detected by periodically checking the router liveness from the clients. If the router does not respond for a certain time period, the clients assume the server crashed. Then the router has to be rebooted.

Whenever a packet is received, a log file is updated storing the time of reception, the counter number, and the correctness of the packet. After each experiment, all the log files produced by the clients are analyzed to check for any possible transmission error.

5. Experimental results

This section presents some statistical result obtained injecting SEU in the router memory. In particular, a single fault at a time has been inserted into a randomly selected bit of each byte of:

- *Code Segment*, whose size is 10,624 bytes
- *Local Data Segment*, which is split into two regions: initialized data (7,316 bytes) and not-initialized data (BSS=Below Stack Segment, 1,860 bytes).

A preliminary analysis of the memory was made and the router's internal variables (Figure 5) classified as:

- *Simple variables* (integer, char, float), used as indexes, accumulator, counter, etc.
- *Data pointer*
- *Functions pointer*.

To assure the validity of the obtained results, each experiment has been repeated four times.

Analyzing the log files created by the clients during the injection experiments, we clustered the behaviors according to the target memory regions:

- *Code Segment*: faults injected in this area produce three different system behaviors:
  - *No-effects*: the system still works properly
  - *Critical error*: the routing functionalities are interrupted, but the Operating System is still active. These kinds of error are not so critical, since it is possible to resume the router in a safe state without rebooting the system (Forward Recovering [11][12])
  - *Crash*: the system has to be rebooted. In this area, we can further distingue between *total crash* (the system stops running in all the experiments) and *partial crash* (the system stops in a subset of the experiments, only).

Table 1 summarizes the analysis of the injections in the Code Segment.

Produced error	Bytes	Percentage
No-effects	6,352	59,8%
Critical errors	1,168	11,0%
Partial crash	2,944	27,7%
Total crash	160	1,5%

Table 1: Analysis of the Code Segment

- *Data Segment*: the faults injected in the local variables of the router produce the following behaviors:
  - *Non-Critical Variables*: the system still works properly and all the packets exchanged are correctly transmitted
  - *Critical Variables*: the router sometime crashes and, thereinafter, all the clients do not receive any packets
  - *Very Critical Variables*: the router always crashes.

Figure 6 shows the geographical distribution of the three different kinds of variables in the memory map, whereas Table 2 summarizes the analysis of the Data Segment.

Looking at the result shown in Figure 5 and Figure 6 one can establish a relationship between the variable class and the behavior of the system when that kind variable is corrupted. Normally the Function Pointers can be classified as Very Critical Variables, the Data Pointer as Critical Variables whereas Simple Variables normally fall in the category of Non-Critical Variables.

Variable effect	Bytes	Percentage
Non-Critical	7,001	76,3%
Critical	211	2,3%
Very Critical	1,964	21,4%

Table 2: Analysis of the Data Segment

6. On-going work

Some software dependability techniques will be applied on the router code in order to enhance the dependability of the router. The statistical study realized in this paper will be used as a benchmark to evaluate the effectiveness of these techniques.

7. Conclusions

In this paper we presented a classification of a router behavior in presence of SEU. The proposed work consists in the design and implementation of a network architecture and of a distributed Fault Injection environment, able to

corrupt the router memory and to observe the fault effects on the distributed system. The network was implemented using an Open-Source router in order to allow direct access to the source code and to use software fault injection techniques.

To get a comprehensive fault behavior classification, experiments were performed both in Data and Code Segments.

8. Acknowledgement

The authors wish to thank Andrea Bardone who implemented the fault injector environment and ran all the injection experiments and Isaac Levendel (Corporate Software Technology Center - Motorola, Inc) for his aid and the innovative advice.

9. Refernces

[1] Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.C., Laprie, J.C., Martins, E., and Powell, D.: Fault Injection for Dependability Validation: A Methodology and Some Applications. IEEE Trans. Software Eng., vol. 16, no. 2, pp. 166-182, February 1990

[2] Segall, Z.: Fiat: Fault-Injection-Based Automated Testing Environment. Proc. 22nd Int'l Symp. Fault-Tolerant Comput., pp. 102-107, June 1988

[3] Kanawati, G.A., and Abraham, J.A.: FERRARI: A tool for the validation of system dependability properties. Proc. 22nd Int. Symp. Fault Tolerant Comput., Boston, MA, USA, July 1992, pp.336-345

[4] Choi, G.S., and Iyer, R.K.: Focus: An Experimental Environment for Fault Sensitivity Analysis. IEEE Trans. Computers, Vol. 41, No. 12, Dec. 1992, pp.1515-1526

[5] Goswami, K.K., and Iyer, R.K.: A Simulation-Based Study of a Triple-Modular Redundant System Using Depend. Proc. Fifth Int'l Conf. Fault-Tolerant Computing Systems, IEEE Press, Picataway, N.J., USA, 1991, pp. 300-311

[6] Clark, J.A., and Pradhan, D.K.: React: A Synthesis and Evaluation Tool for Fault-Tolerant Multiprocessor Architecture. Proc. 1993 Annual Reliability and Maintainability Symp., IEEE Press, Picataway, N.J., USA, 1993, pp. 428-435

[7] Linux Web Site: http://www.linux.org/

[8] RFC 768 (UDP): https://www.faqs.org/rfc/rfc768.html

[9] http://tvdg10.phy.bnl.gov/seutest.html

[10] http://www.research.ibm.com/journal/rd/ziegl/Ziegler.html

[11] Pradhan, D.K Fault-Tolerant Computer System Design, Prentice Hall PTR 1995.

[12] Jalote, P. Fault-Tolerance in Distributed Systems, Prentice Hall PTR 1994.

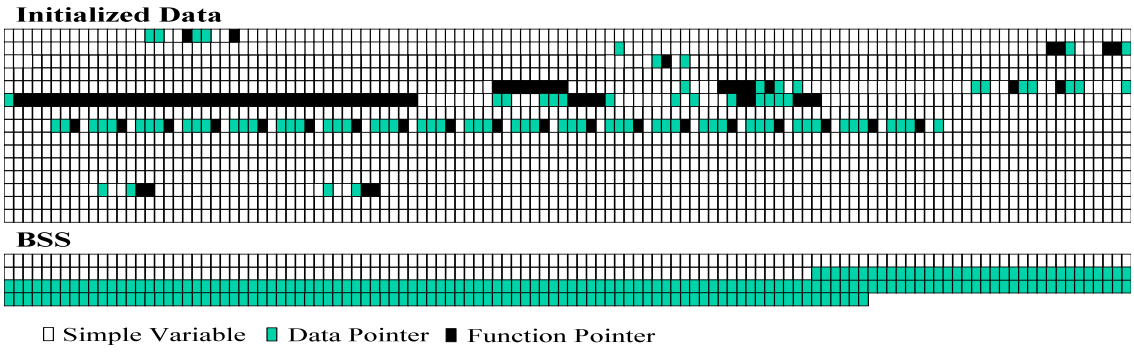

Figure 5: Variable distribution in the router's memory

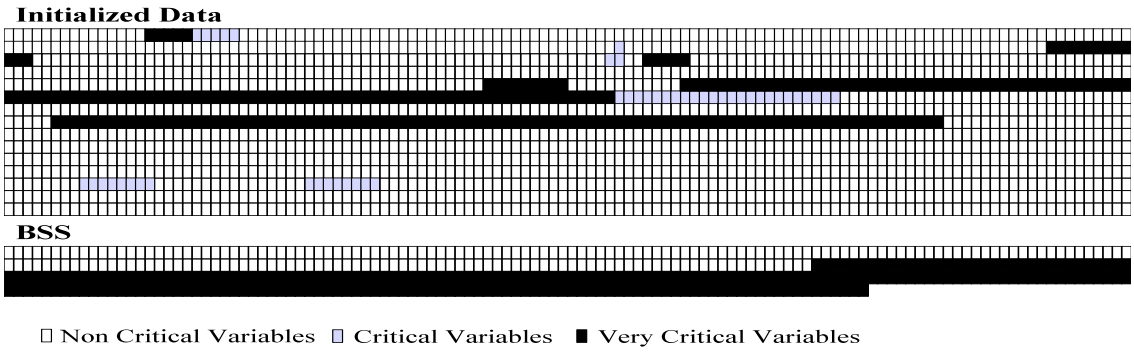

Figure 6: Fault effects on the router's memory