DOCTOR: An IntegrateD SOftware Fault InjeCTiOn EnviRonment — An Extended Abstract*

Seungjae Han, Harold A. Rosenberg, and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122.
email: {sjhan, rosen, kgshin}@eecs.umich.edu

The increasing complexity of contemporary computer architectures and the high-degree integration of functions into an ever-shrinking VLSI chip have made it difficult to evaluate/validate computer systems with hardware fault injections. Especially, the limitation of fault-injection location to pin boundaries results in lack of control over high-level fault manifestations or errors. Since our early work on hardware fault injection in [1, 2], we have been building a dependability evaluation environment based on software fault injection [3].

Our research into fault injection has taken a three-pronged approach. First, we have enhanced the capability of representing diverse fault type, because most software-implemented fault injectors are limited to specific fault types. A more sophisticated fault model than the commonly-used, basic memory fault model is required to fully emulate the effects of various faults. Considering the trend that system-level components are becoming the basic units of fault confinement, diagnosis, and replacement, we have decided to support three classes of faults: memory faults, communication faults, and processor faults. Our fault model also supports three temporal types of faults: transient, intermittent, and permanent. Various types of probability distributions are also provided to specify fault inter-arrival times. A memory fault can be injected as a single bit, two-bit compensating, whole byte, or burst (of multiple bytes) error. The symbol-table information can be used to decide injection locations, and the contents of memory at the selected address are partially or totally set, reset, or toggled. Communication faults affect the delivery of messages. Messages can be lost, altered, or delayed. If the node has multiple incoming and outgoing links, as in a point-to-point architecture, different fault types can be specified separately for each link. Several options are provided to allow the injection of a variety of failure semantics, including Byzantine failures. To emulate the effect of faults in a processor, we make use of executable image modification, in which instructions generated by the compiler are modified or extra instructions are inserted for a fault-injection purpose. Currently, the alteration of control flow, register errors, and ALU errors are supported.

The second is to provide a complete set of tools for automated fault-injection experiments. Generally, a fault-injection experiment environment consists of 4 major components: the target system with fault-tolerance mechanisms which are to be evaluated, a fault injector, the workloads to be run on the target system, and a dependability monitoring tool. The first target system of DOCTOR is HARTS, a real-time distributed system [4, 5]. The Software Fault Injector (SFI), the core part of DOCTOR, supports the fault models described above, and consists of three modules: the Experiment Generation Module (EGM), the Fault Injection Agent (FIA), and the Experiment Control Module (ECM). EGM accepts an experiment description file which contains the experiment plan and the information about the fault type and injection timing. EGM is responsible for generating the executable images of workloads which will be down-

* The work reported here is supported in part by the Office of Naval Research under Grants N00014-91-J-1115 and N00014-92-1080, the National Aeronautic and Space Administration under Grant NAG-1493, and the National Science Foundation under Grants MIP-9012549 and MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

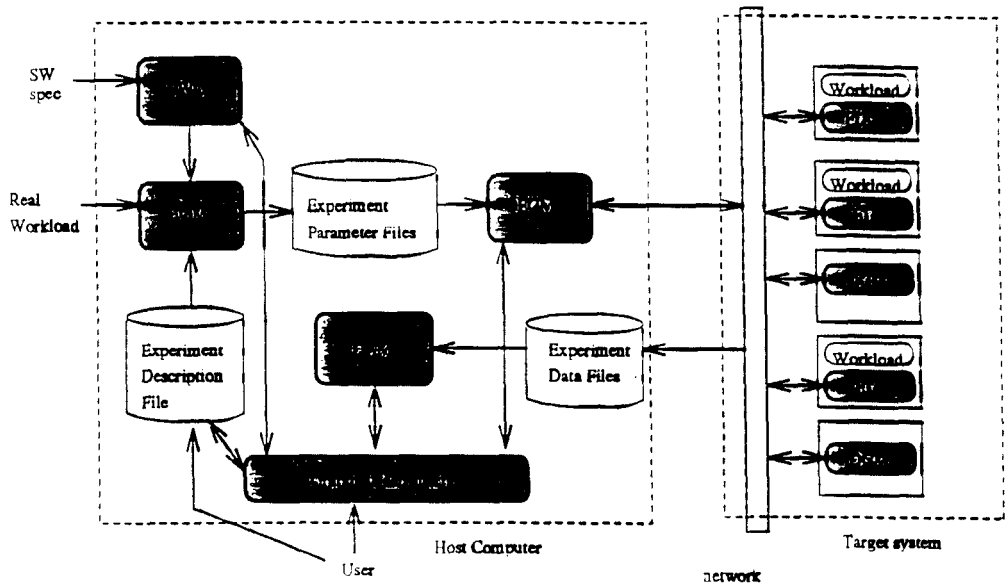

Figure 1: The organization of DOCTOR

loaded (from the host) to the target system. When the workloads are compiled, the symbol-table information is extracted, and some library programs are attached to workload executable code. Necessary executable code modification for injecting processor faults are also done here. FIA is a separate process which runs on the same node as the workload. It receives control commands from ECM via a communication network and executes them such as enabling faults or making workloads wait/start/stop. ECM functions as a supervisor. It controls all other modules, and sets up an experiment environment. It uses the experiment parameter files generated by EGM for each node involved in an experiment in order to generate proper commands for FIA. ECM synchronizes the start/end of each runs among several nodes, and performs required re-initialization steps for user-transparent automated multi-run facility. In most of the fault-injection work reported in the literature, only a couple of real programs are used as workloads. A workload produces demands for the system resources, so the structure and behavior of the workload may affect the dependability of fault-tolerance mechanisms under test significantly. We provide a Synthetic Workload Generator(SWG) [6], so that experiments can be conducted under systematically-controlled workload conditions. The dependability monitoring tool of DOCTOR is composed of two modules: Data Collection Module (DCM), Data Analysis Module (DAM). DCM collects event data which are produced by FIAs or fault-tolerance mechanisms under test, such as fault injection/detection events. Basically, it is a expansion of HMON [7], which provides non-intrusive monitoring of system performance. DAM analyzes the data collected by DCM. For example, it calculates the fault detection coverage and latency. Since DAM has a modular structure, other analysis capabilities can be added easily. DAM has a graphic display function to present analysis results. In addition, a comprehensive graphic user interface helps the user design and control experiments in an X window environment. Figure 1 shows the organization of DOCTOR.

The third focus is on the consideration of portability. All previously-implemented SFI methods have a common restriction that they were developed for specific target systems. That is, portability — an important merit of SFI — has not been figured into their design. By minimizing its dependence on the underlying hardware architecture and operating system, a SFI tool which runs on one system can be ported to another with minimal effort. Fault-injection experiments can then be performed during early design phases without developing a new fault injector for each target system. DOCTOR does not rely on any particular system, though our main interest lies in distributed real-time systems. However, complete independence of a SFI tool from hardware and system software is impossible to accomplish, since in many cases the fault-tolerance features under test are implemented at the hardware or system software level. A realistic way to reduce the porting effort is to utilize "standard" features available in commonly-used

systems, and to separate the system-dependent parts from the rest.

We have demonstrated the capability of DOCTOR through extensive experiments, such as evaluating the effectiveness and the associated performance overhead of error detection and diagnosis methods by injecting various types of faults tens of thousands of times. Detailed specifications and result data of experiments can be found in [8]. To demonstrate the portability of DOCTOR, we will port this tool set to additional distributed systems, and evaluate the required amount of effort. We also plan to measure the fault injection and data collection overhead of our tools, and will explore new methods of reducing and controlling this overhead. In addition, we are currently exploring the issues involved in formalizing both the specification of fault-injection experiments, and the systematic selection of the faults to be injected.

References

[1] K. G. Shin and Y. H. Lee, "Measurement and application of fault latency," *IEEE Trans. Computers*, vol. C-35, no. 4, pp. 370–375, April 1986.

[2] M. Woodbury and K. G. Shin, "Measurement and analysis of workload effects on fault latency in real-time systems," *IEEE Trans. Software Engineering*, vol. 16, no. 2, pp. 212–215, 1990.

[3] H. Rosenberg and K. G. Shin, "Software fault injection and its application in distributed systems," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 208–217. IEEE, 1993.

[4] K. G. Shin, "HARTS: A distributed real-time architecture," *IEEE Computer*, vol. 24, no. 5, pp. 25–35, May 1991.

[5] K. G. Shin, D. Kandlur, D. Kiskis, P. Dodd, H. Rosenberg, and A. Indiresan, "A distributed real-time operating system," *IEEE Software*, pp. 58–68, September 1992.

[6] D. L. Kiskis, *Generation of Synthetic Workloads for Distributed Real-Time Computing Systems*, PhD thesis, University of Michigan, August 1992.

[7] P. S. Dodd and C. V. Ravishankar, "Monitoring and debugging distributed real-time programs," *Software–Practice and Experience*, vol. 22, no. 10, pp. 863–877, October 1992.

[8] S. Han, H. A. Rosenberg, and K. G. Shin, "Doctor: An integrated software fault injection environment," CSE Technical Report, CSE-TR-192-93, EECS Department, University of Michigan, December 1993.