Fault-Tolerant Computer System Design
ECE 60872/CS 59000

Topic 9: Validation

Saurabh Bagchi
ECE/CS
Purdue University

Outline

- Introduction
- Validation methods
- Design phase
  - Fault simulation
- Prototype phase
  - HW or SW implemented fault injection
-

Challenges

- **Assessing the system dependability for**
  - different technologies
  - different computers
  - different network topologies
  - different communication protocols
- **Validating the networked system characteristics**
  - dependability: (i) user level, (ii) system level, and (iii) network level
  - availability validation: (i) detection and (ii) recovery

Experimental Analysis
Different Design Phases

Early Design Phase

Approach and Goals:

- CAD environments used to evaluate design via simulation
- Simulated fault injection experiments
- Evaluate effectiveness of fault-tolerant mechanisms
- Provide timely feedback to system designers

Information produced

fault models, error latency, error detection coverage, recovery time distribution

Limitation:

Simulations need accurate inputs and validation of results

Prototype Phase

Approach and Goals:

- System run under controlled workload conditions
- Controlled fault injections used to evaluate system in presents of faults

Information produced

error latency, propagation detection distributions, availability

Limitation:

Injected faults should create/induce failure scenarios representative of actual system operation

Operational Phase

Approach and Goal:

- Measurement-based approach to study naturally occurring errors
- Study systems in the field, under real workloads
- Analyze collected error and performance data
- Provide information on actual failure characteristics

Information produced

actual failure characteristics, failure rates, time to failure distribution

Limitation:

Reported errors are representative of the systems studied

Evaluation - Experimental Methods

- **Design phase**
  - Fault simulation (simulated fault injection)
  - Electrical, logic, or functional level
  - Hierarchical simulation
- ***Issues:*** simulation time, level of simulation, fault conditions, accurate fault models
- **Prototype phase**
  - Fault injection in prototype systems
  - Hardware fault injection
  - Software fault injection
  - Radiation-based fault injection
- ***Issues:*** Fault models, joint HW /SW fault injection, injection into networked applications
- **Operational phase**
  - Study of naturally occurring faults in real environments
  - Essential for believable analysis of today's complex systems
  - What can we say about future systems based on measurements from current systems
- ***Issues:*** HW/SW instrumentation, analysis tools

Key Issues in Fault Injection

- Effective fault injection mechanisms using hardware, software, and hybrid technology to accurately assess and validate networked systems
- Practical evaluation methods to accurately quantify fault effect and recovery mechanisms in complex environments
- Evaluation of error detection, diagnosis, and recovery techniques
- Quantification of confidence in the fault-injection based validation
- Usable fault tolerance benchmark for assessing systems and networks
- Common evaluation/validation framework

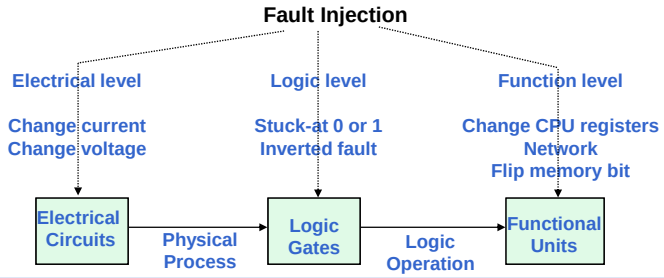
Metrics of Fault Injection

- *Fault activation* refers to the activation or access of injected faults.
  - For example, fault injected into a register is activated when that register is read before the register is overwritten.
- *Fault activation level* refers to the ratio of the number of activated faults (F_a) to the total number of injected faults (F_i). Therefore, *fault activation level* = F_a/F_i
- Metrics for a fault tolerant technique
  - Accuracy/Recall = (Number of activated faults that were dealt with)/(Total number of activated faults)
  - Precision = (Number of cases out of the denominator that were actual activated faults)/(Number of cases that were dealt with by the fault-tolerant technique)

Design Phase: Simulation at Different Levels

- **Electrical level**
  - **transistor → circuit → chip**
- **Logic level**
  - **circuit → VLSI systems**
- **Function level**
  - **VLSI system → computer and network systems**

Levels of Simulated Fault Injection

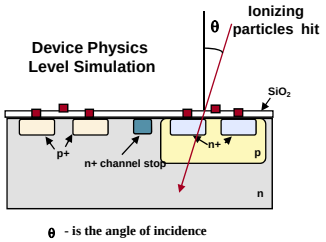


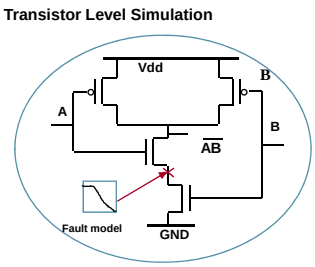
Questions to Ask In Simulated Fault Injection

- In simulation-based fault injection, the whole system behavior is modeled and faults are injected to the simulated model of the systems
- This technique is often applied in the design phases to allow test and validation of error handling techniques before a physical prototype is available
- **Fault models**
  - Fault conditions, fault types, number of faults, fault times, fault locations
- **Workload**
  - Real applications, benchmarks, synthetic programs
- **Simulation time explosion**
  - Mix-mode simulation, importance sampling, concurrent simulation, accelerated fault simulation, hierarchical simulation

Simulated Fault Injection at Electrical Level

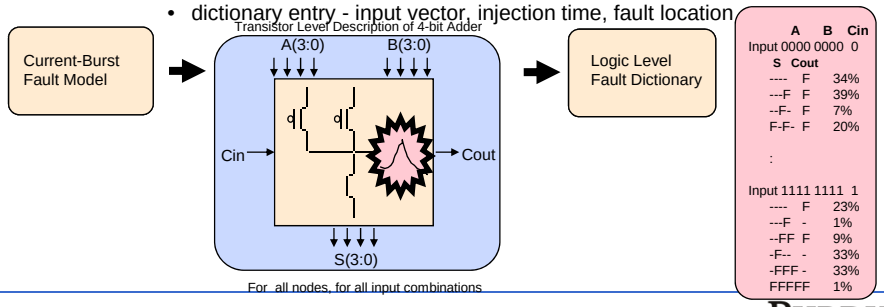
- Why is it needed?
  - Study the impact of physical causes
  - Simple stuck-at models do not represent many real types of faults





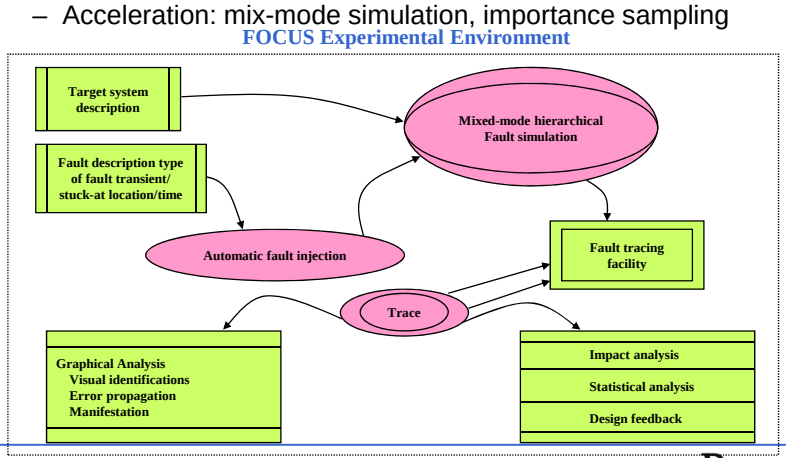
Simulated Fault Injection at Logic Level

- **Fault Mode**
  - ***Basic models***
    - stuck-at (permanent) - forcing logic value for entire simulation duration
    - inverted fault (transient) - altering logic value momentarily
  - ***Fault dictionary approach***
    - Use electrical level simulation to derive logic-level fault models
    - dictionary entry - input vector, injection time, fault location



Simulated Fault Injection at Chip Level

- FOCUS & SPLICE!
  - A chip-level simulation environment
  - Acceleration: mix-mode simulation, importance sampling

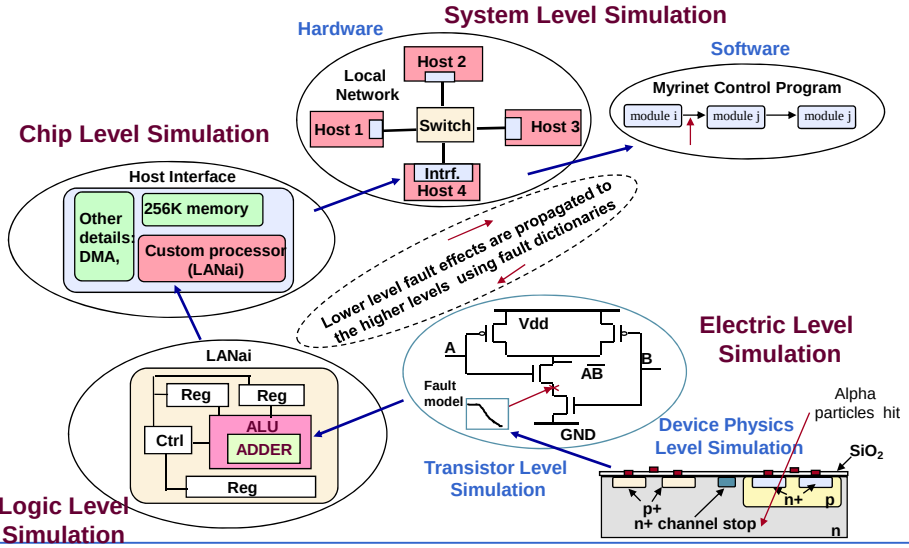


Simulated Fault Injection at Function Level

- Diversity of Components
  - Object-oriented approach
- Fault Models
  - Various types - depending on the type of components
  - Examples
    - Single bit-flip for a memory or register fault
    - Message corruption for communication channel fault
    - Service interrupt for a node fault
  - More detailed fault models derived from lower-level simulation
- Impact of Software
  - Impact of faults is application dependent
  - Software effect can be studied at this level

Hierarchical Fault Simulation (Reference [1])

Example: LANai Processor of Myrinet Network Switch



Prototype Phase: Software Implemented Fault Injection

- ***Advantages*:** flexibility, low cost
- ***Disadvantages*:** perturbation to workload, low time resolution

Targets for software fault injection

- Software faults and errors
  - modify the text/data segment of the program
- Memory faults
  - flip memory bits
- CPU faults
  - modify CPU registers, cache, buffers
- Bus faults
  - use traps before and after an instruction to change the code or data used by the instruction and then restore them after the instruction is executed
- Network faults
  - modify or delete transmitted messages
  - introduce faults in network controllers, drivers, buffers

Characteristics of SWIFI

Table 1: Characteristics of Software Fault Injection Techniques

Software Fault Injection Hierarchy				Main Property	Fault Type	Advantage	Disadvantage
Compile-Time	Software Mutation	Source Code Mutation		A modified version of high-level source code is loaded to emulate common practical fault patterns	Mutation	More close to emulate real software mistakes/bugs; Suitable to emulate permanent faults.	Case specific. Require manual intervention; Hard to emulate transient faults; Need to recompile
		Bytecode Mutation		For JAVA; C#; In bytecode level	Bit flips, Mutation	No source code needs to be modified.	Additional tools are needed to transform the bytecode
Runtime	Time-Trigger			Software/hardware Interrupt; Random	Bit flips, mutation	Random; Can be combined with other type of injections	Hard to re-create the faulted scenario (Low reproducibility)
	Mutation	Interface (Parameter)		API; syscall; Parameter in; Function return value	Mutation	Good Portability; Can be implemented to be automatically run.	Limited area of operating systems are tested.
		Code; bytecode		Emulation including performance faults	Mutation	Match faults in reality; Inject to virtual machine's object attribute values, methods parameters	Case specific; Hard to be run automatically
	User-Space Ptrace (Breakpoint)	TRAP instruction		Activation; Monitor; Code, Register; Crash latency	Bit flips	Multiple breakpoints; Automatic experiments; Emulate transient/ permanent faults	Cannot monitor data access; Target process needs to agree to be traced.
		Debug register		Activation; Monitor; Code, Data, Stack, Register; Crash latency	Bit flips	Inject to code/data/stack parts; Automatic experiments; Emulate transient/ permanent faults	Not all kernels support debug register access in Ptrace.

Characteristics of SWIFI (Continued)

Runtime	Kernel-Space (Breakpoint)	Kernel	Activation; Monitor; Code, Data, Stack, Register; Crash latency	Bit flips	Use Debugging and Performance features; Automation; Emulate transient/ permanent faults	Need to have knowledge of kernel
		User Virtual Address	Virtual address injection; Activation; Monitor; Code, Data, Stack, Register; Crash latency	Bit flips	No need to use Ptrace; Minimum inference to applications; Automatic experiments; Emulate transient/ permanent faults	Need to use additional code to differentiate overlapped virtual address space for different processes
	FPGA	Configuration bit file Code Mutations	Dynamic Reconfigurable FPGA;	Bit flips	Runtime directly injects to FPGA; Evaluate hardware prototypes	Hard to monitor behavior after injections; Need to have knowledge of FPGA; Need to re-compute checksum
	Security Threat		Penetration Testing	Mutation, Bit flips	Fast and cheap, effective	Front impact testing, inefficient
Simulation	Simulation-related		Processor level; Combinational logic; Flip-flops	Bit flips	Easy to monitor system behavior in presence of faults;	Results are based on the design of simulator;

Difficulties in using SWIFI Tools

- Often fault injection is needed to answer the following questions
  - Comparative studies – is my app more reliable on Android or iOS?
  - Dependability benchmarking – rank the various flavors of Linux by their reliability
  - System-level evaluation of large systems – comprising heterogeneous components
  - Special-purpose embedded systems – which have resource constraints
- To answer these questions, the SWIFI tool needs to be ported to different platforms
- Porting effort is high

NFTAPE (Reference [3])

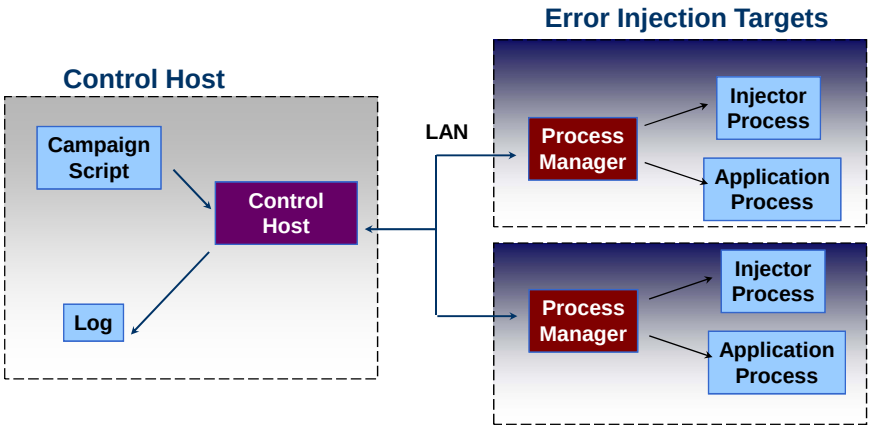
NFTAPE is a tool for conducting automated fault/error injection-based dependability characterization

- *Tool*, which enables a user: (1) to specify a fault/error injection plan, (2) to carry on injection experiments, and (3) to collect the experimental results for analysis.
- Facilitates *automated* execution of fault/error injection experiments.
- Targets assessment of a broad set of *dependability metrics,* e.g., availability, reliability, coverage, mean time to failure.
- Operates in a *distributed environment*.
- Imposes minimal disturbance of target systems

NFTAPE Approach to Fault/Error Injection

- Lightweight Fault/Error Injectors (LWFIs)
  - Small (in terms of a code size) entities responsible for injecting faults/errors
  - Rely on other processes for services such as fault triggering, data logging, etc.
  - Example LWFIs: driver-based, target-specific
- API for implementing and invoking Triggers, LWFIs, and support programs
  - Less work to create components necessary for conducting fault/error injection experiments
  - Components conform to standard interface
  - Components can be reconfigured using the standard interface (e.g., swap triggers)
- Reusability
  - LWFIs and triggers collected in a library
  - Same configuration, startup, and logging process for different fault/error injection campaigns
  - One learning curve

NFTAPE Architecture



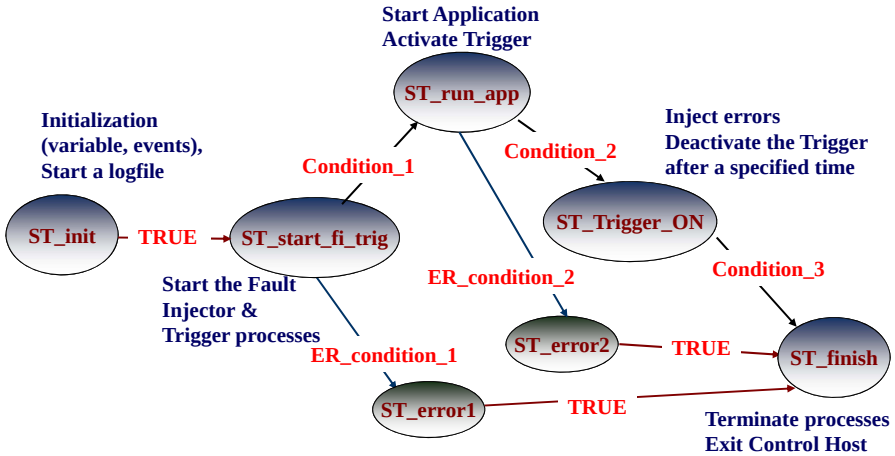
Control Host & Process Manager

- Control Host
  - Processes a Campaign Script, a file that specifies a *state machine* or control flow followed by the control host during the fault injection campaign
  - Simple yet general way to customize a fault injection experiment
  - Experiments controlled by the Common Control Mechanism
  - Implemented in Java to ensure portability
- Process Manager
  - Daemon on each target node to manage processes on the target node(s) including process execution and termination
    - processes include: injectors, workloads, applications, monitors
    - all processes are treated the same as an *abstract process object* rather than a process of some specific type
  - Facilitates communication between the Control Host and Target Nodes.

Fault Injectors - examples

- *Driver-based* injector - uses dedicated device driver to inject memory, registers, OS functions, I/O devices (e.g., Linux, Solaris)
- *Ptrace-based* injector (e.g., Linux, Lynx) - controlled injection to the target process memory and registers;
- *Proc file based injector* (e.g., Solaris) – controlled injection to the memory image of a process
- *Network injector* - employs dedicated software for controlling network hardware to inject faults into network cards/controllers; can intercept and corrupt messages
- Use of performance monitors (built into the CPU) to trigger fault injection
- *Injection of delay faults/errors* – controlled, temporary suspending of interrupts

State Machine of an Example Campaign Script



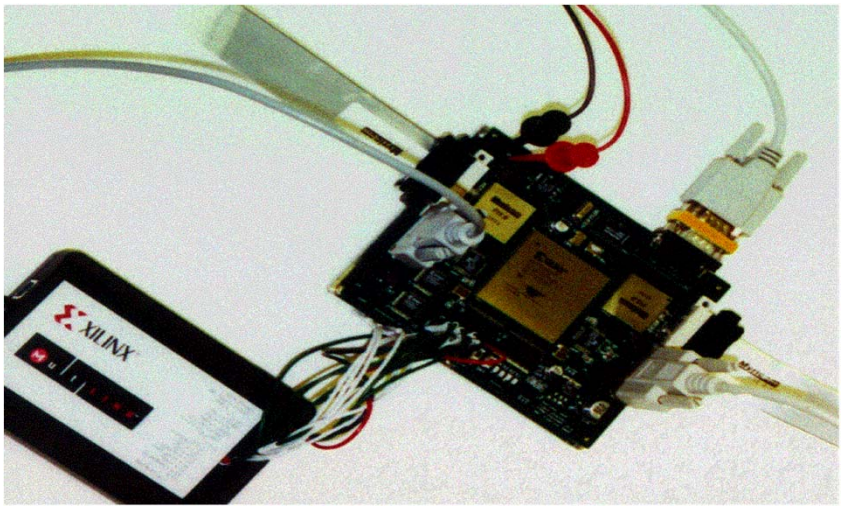
Hardware Fault Injector for Inducing Network Faults

- A versatile device for supporting injection of random and specific faults in a network environment.
- Design based on FPGA as the core structural component, which allows the device to be programmed to:
  - accept configuration commands generated either internally (i.e., by the device itself) or by an external system
  - provide other services e.g., data monitoring, statistics gathering, or prototyping new designs
- Design enables precise synchronization of fault injection hardware with target systems – Myrinet network and Fiber Channel - while running at-speed of the network
- Fault injector inserted in the data path (i.e., in the network) to decode the data patterns, modify the data if necessary (i.e., inject faults), and then retransmit the data (i.e., send the corrupted/modified data to the network).

Capabilities of HW Fault Injection Techniques
Summary

Injection method	Synchronization with the system activity	Injection at the system speed	Targeting faults to specific locations	Guarantees of fault activation	Example tools and references
Pin-level at the signal level; not signal partitioning	Speculative	Yes	Yes	Low speed	Messaline [Arl89], Hybrid (signal-level & SWIFI) [Kan95]
Pin-level by signal interception; Signal partition by inserting the injection logic into the path of the signal	Full	Yes (if the delay induced by the inserted logic does not exceed that allow by the system)	Yes	Yes	Messaline [Arl89], Rifle[Mad94]
Heavy Ion Radiation	Speculative	Yes	No	No	[Kar94]
Power supply disturbances	Speculative	Yes	No	No	[Mir95]
Electromagnetic interferences	Speculative	Yes	No	No	[Kar97]
Laser (LFI)	Speculative	Yes	Yes	No	[Sam97]
Built-in logic	Full	No	Yes	Yes	FIMBUL [Fol98]]

Assembled Fault Injector



Conclusion

- Fault injection is a means to effectively test and stress fault tolerance mechanisms before they are deployed in the field on an actual system
- Fault injection can be done through hardware or software
- Software-implemented fault injection (SWIFI) is the dominant mode of system verification, especially for complex software systems
- It is common to compare multiple systems (or versions of the same system) using dependability attributes collected from a fault injection campaign

References

1. Stott, D. T., Ries, G., Hsueh, M. C., & Iyer, R. K. (1998). Dependability analysis of a high-speed network using software-implemented fault injection and simulated fault injection. Computers, IEEE Transactions on, 47(1), 108-119.
2. Z. Kalbarczyk, R. K. Iyer, G.L. Ries, J.U. Patel, M.S. Lee, and Y. Xiao “Hierarchical Simulation Approach to Accurate Fault Modeling for System Dependability Evaluation”, IEEE Transactions on Software Engineering, vol. 25, no.5, September/October 1999, pp.619-632.
3. Stott, D.T.; Floering, B.; Burke, D.; Kalbarczpk, Z.; Iyer, R.K., "NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors," Computer Performance and Dependability Symposium, 2000. IPDS 2000. Proceedings. IEEE International , vol., no., pp.91-100, 2000