Measuring Application Error Rates for Network Processors

A. Mallik, M. C. Wildrick, G. Memik
Department of Electrical and Computer Engineering
Northwestern University
2145 Sheridan Road
Evanston, IL 60208

Abstract- Faults in computer systems can occur due to a variety of reasons. In many systems, an error has a binary effect, i.e. the output is either correct or it is incorrect. However, networking applications exhibit different properties. For example, although a portion of the code behaves incorrectly due to a fault, the application can still work correctly. Integrity of a network system is often unchanged during faults. Therefore, measuring the effects of faults on the network processor applications require new measurement metrics to be developed. In this paper, we highlight essential application properties and data structures that can be used to measure the error behavior of network processors. Using these metrics, we study the error behavior of seven representative networking applications under different cache access fault probabilities.

I. INTRODUCTION

Introduction of a fault into the system is an inherent possibility of all computer processors. The sources for these faults can be different. They may arise from adverse environmental conditions [19], physical hardware defects, or from electronic noise, incorrect device utilization, or logical design flaws [9]. Different kind of faults affects the system in different ways. Additionally, modern processors are advocating for aggressive scaling of the supply voltages (V_{dd}) which increases the probability of a fault occurrence. Increasing clock rates, and the use of flip-chip packaging is expected to introduce complex problems in this field. While it is critical to put every effort to avoid these faults by careful circuit design and packaging, they can still occur and need to be addressed.

The effect a hardware fault has on a system is application dependant. When we are designing a desktop processor or a server, ignoring faults is not an option. A user won’t accept a computer which can crash anytime or give wrong results due to a fault. Hence the processor should be designed to capture and eliminate faults. The user expects a desktop processor or server to work continuously for days or weeks. In such cases, hardware faults cannot be tolerated. However, for networking applications—a certain level of error is acceptable. We can allow some errors to happen due to hardware faults as long as the integrity of the system’s behavior can be maintained. This is also related to the properties of the systems: networking software/systems are implemented with the assumption that the hardware can fail (e.g. routers can drop packets). Therefore, faults at a certain level are acceptable for such processors.

Under the presence of faults, even if the system integrity is unchanged, it affects the operation of the system. As a result, the system will operate differently depending on what kind of data becomes corrupted. For instance, in the presence of electronic noise a single piece of transient data may get corrupted. This affects behavior only momentarily. On the other hand, a static data element might be damaged—such as a lookup table which is used for every packet process in a network system. This would the system for a longer period of time. Additionally, recovering procedure and time for such errors would be more difficult.

In this paper the susceptibility to fault and the resulting behavior is analyzed for Network Processors (NPs). Particularly, we study several applications from the NetBench benchmarking suite [12] and define error metrics for each of these applications. We also perform a study where we introduce cache faults and measure its effect on these applications. NetBench suite consists of a variety of applications that can be used to simulate several of a network processor’s functions. Among these are routing, encryption, and packet filtering, all of which behave differently in the presence of faults.

Traditionally, embedded processors in networks are either custom-designed ASIC chips or variations of general-purpose processors. Both schemes have their advantages and disadvantages. ASIC chips have better performance, but they have higher manufacturing costs and lack the flexibility of programmable processors. If there is a change in the protocol or application, it is hard to reflect the change in the ASIC design. General-purpose processors, on the other hand, are not optimized for networking applications and hence do not provide satisfactory performance for most of the applications.

Network Processors eliminate the drawbacks of general-purpose processors and ASIC designs by combining the flexibility of general-purpose programmable processors and performance of ASIC chips.

We have examined and classified different kinds of errors that may occur in a network system. One type is marked as a volatile error, affecting data only temporarily, perhaps an incorrectly received network packet or the corruption of a temporary value. In general this type of error will only concern a limited amount of data, and will not noticeably affect performance provided that the error does not continually reoccur. While processing a network application we can ignore a volatile error whose occurrence is limited to very few packets. The other type is a nonvolatile error, affects on a static data structure such as the routing table used in a NAT application. This type of error will have a permanent affect on the system. As we would use the control structures for the processing of each packet, a fault in that part may corrupt many calculations over time before it is corrected. Our goal in this paper is to define data segments of these applications that can be used to measure their error behavior.

The rest of the paper is organized as follows. In the next section, we overview the networking applications and present the error measurements we have defined for each application. Section III discusses our simulation environment. Section IV presents the simulations results. Section V describes the related work and Section VI concludes the paper with a summary.
II. APPLICATIONS AND ERROR MEASUREMENT

In this section, we discuss the networking applications studied in this paper and present the data structures used to measure application errors for each benchmark program. We selected seven applications from the NetBench [12] suite. The applications are listed in Table I. NetBench is a benchmarking suite designed for NPs. It contains applications representing level 3 tasks (e.g. route) as well as higher-level programs (e.g. MD5).

For each network application, important data structures and output of key function units are identified. Our goal is to make a comparison of these data values between the correct execution and a faulty execution. Using simulation results, we can calculate the statistical probability of an error to happen in the application. We can notice that a part of these data structures have more impact on the overall output than others (e.g. a routing table error is more important than an error in the ttl value calculation). Although all of them are classified as Error Keys, some of them react more sensitively towards a fault. However, no weights are assigned for each error metric. In this analysis, we simply list the structures that are important in the execution. Once we have marked the Error Keys, we effect cache faults on the structures in Section IV. It must be noted in this regard that there is an assumption from the user’s part, about the completion of an application even during high probability of fault occurrence. In reality, execution of an erroneous code doesn’t guarantee completion.

As the code may read erroneous data, it may turn into an infinite loop or may try to get access to some non-existent data during execution. Such events would cause the system to crash. This is a possible outcome for each of the applications we are investigating and is of interest to us for measuring the effects of faults. We classify such errors which prevents the infinite loop or may try to get access to some non-existent data during execution. Such events would cause the system to crash. This is a possible outcome for each of the applications we are investigating and is of interest to us for measuring the effects of faults. We classify such errors which prevents the program to continue its execution as a fatal error. In Section IV, we also present the probability of a fatal error for different fault rates. In the following, we list the selected application followed by the application-level error metric used to measure the effects of faults.

**CRC:** The CRC-32 checksum calculates a checksum based on a cyclic redundancy check as described in ISO 3309 [10]. CRC-32 is used in Ethernet and ATM Adaptation Layer 5 (AAL-5) checksum calculation. The code is available in the public domain [5]. The errors are measured using two data structures: the crc table and the crc accumulator value calculated for each packet. Note that, the errors in the crc table are more important as they would affect multiple packets during the processing. Any error on the accumulator calculation part would concern only one packet.

**TL:** TL is the table lookup routine common to all routing processes. We have used radix-tree routing table, which was used in several UNIX systems. The code segment is from FreeBSD operating system [8]. The error metrics in the TL application are: the radix tree nodes traversed and the RouteTable entry for each packet.

**ROUTE:** IPv4 routing according to RFC 1812 [2] is implemented in the Route application. When a packet arrives in a router its next network hop is decided by the router. Route implements the table lookup along with internet checksum (for the header). During processing, there are changes in the header (for example, the Time-To-Live value). It may fragments the packet and forwards it. The code is from the FreeBSD operating system [8]. The values observed in the route application are: the entries in the created RouteTable, the checksum value, the ttl value, and the radix tree entries traversed for each packet.

**DRR:** Deficit-round robin (DRR) scheduling [18] is a scheduling method implemented in modern network switches. In DRR, all the connections through the router have separate queues. Using these queues, the router tries to accomplish a fair scheduling by allowing same amount of data to be passed from each queue. The implementation is based on the algorithm by Shreedhar and Varghese [18]. The data values in the DRR application are: the entries in the created RouteTable, the radix tree entries traversed for each packet, the value of the deficit list for each packet, and the deficit information read for the packet.

**NAT:** Network Address Translation (NAT) is a common method for IP address management. NAT operates on a router, usually connecting two networks, and translates the private (not globally unique) addresses in the internal network into legal addresses before packets are forwarded onto the public network. Hence, for any departing packet, the source IP on the packet should be changed. Similarly, the destination address on any incoming packet should also be modified. The program accomplishing this task is using several routines from FreeBSD operating system [8]. The data values used for measuring errors in nat are: initial IP source address, value in the interface for translation, translated IP source address, the IP destination address after translation, the entries in the NAT table, and the radix tree entries traversed for each packet.

**MD5:** Message Digest algorithm (MD5) creates a signature for each outgoing packet, which is checked at the destination [15]. The signature is cryptographically secure, hence if the received packet does not match the signature, then the receiver will assume that the packet is unreliable and discard it. The implementation is from RSA Data Security, Inc. [16]. The errors in MD5 are binary errors. So they are easily detectable. In other words, if the output string of the erroneous execution does not exactly match the correct execution, the packet is said to be processed incorrectly. We then measure the fraction of packets incorrectly processed.

**URL:** URL implements URL-based destination switching, which is a commonly used content-based load balancing mechanism. In URL-based switching, all the incoming packets to a switch are parsed and forwarded according to URL. For example, all image requests might be sent to an image server. This application increases the utility of specialized servers in a server farm. The implementation is based on the description from PMC-Sierra [14]. The data structures in the URL application that are observed are: URL table entries, final IP destination address, RouteTable entries, the checksum value, the ttl value, and the radix tree entries traversed for each packet.

<table>
<thead>
<tr>
<th>Application</th>
<th>Arguments</th>
<th>Number of inst. simulated [M]</th>
<th>Number of cache acc. [M]</th>
</tr>
</thead>
<tbody>
<tr>
<td>crc</td>
<td>5000</td>
<td>145.8</td>
<td>59.8</td>
</tr>
<tr>
<td>tl</td>
<td>128 5000</td>
<td>6.9</td>
<td>3.9</td>
</tr>
<tr>
<td>route</td>
<td>128 5000</td>
<td>14.2</td>
<td>7.1</td>
</tr>
<tr>
<td>drr</td>
<td>128 5000</td>
<td>12.9</td>
<td>7.9</td>
</tr>
<tr>
<td>nat</td>
<td>128 5000</td>
<td>11.4</td>
<td>5.6</td>
</tr>
<tr>
<td>md5</td>
<td>5000</td>
<td>209.1</td>
<td>73.2</td>
</tr>
<tr>
<td>url</td>
<td>small_inputs 5000</td>
<td>497.0</td>
<td>249.1</td>
</tr>
</tbody>
</table>
III. SIMULATION ENVIRONMENT

We use the SimpleScalar/ARM [4] for our simulations. We modified the input set to model a processor similar to execution cores in a variety of Network Processor architectures. Particularly, we simulate a processor similar to StrongARM 110 with 4 KB, direct-mapped L1 data and instruction caches with 32-byte line-size, and a 128 KB, 4-way set-associative unified L2 cache with a 128-byte line-size. We first modified the applications to output the values of data structures mentioned in the previous section. Then, we have modified the simulator to introduce random errors into the execution and to simulate the effects of the introduced errors. We chose an initial error probability of $12\times10^{-8}$ per bit, as reported by Shivakumar et al. [17]. The error rate is calculated for each bit accessed independently. Therefore, we do not simulate the effect of relation between errors. Then, we increase the error rate in steps until it is set to $819.2\times10^{-8}$.

IV. SIMULATION RESULTS

This section describes the simulation results observed for selected Network applications. For different error rates, effect on the data structure of each application discussed in section II are recorded.

Each of the application can sustain the introduced error to varying extent. For smaller error rates we observed the execution of the application without any observable error in the data structures and the application output. With increase in the error introduction rate, the applications started to produce erroneous outputs and data structures. For larger error rates, the overall data integrity of the system was lost and the applications crashed. This is indicated by the fatal error probability.

Figure 1 to 7 describes the behavior of seven selected network applications for different error introduction rate.
High transient-error tolerant System Design has traditionally been considered in the context of systems that operate in high-radiation environments or in outer space, where there is a heavy concentration of alpha-particles and atmospheric neutrons [20]. Recent IBM Research showed that computer systems are susceptible to transient faults induced by these particles [19]. In circuit verification area, there has been a strong emphasis on reliability which is an important problem in IC fabrication. Earlier research have studied potential errors in the pre-silicon [3] stage and also subsequent to the fabrication process [1]. High transient fault resilient computer systems design [13] has gained greater significance due to the combined effect of higher integration densities, lower voltages, and faster clock frequencies.

Fault injection is an attractive method for validation for estimating the dependability of computer systems [11]. Studies have already shown that the workload has a significant effect on the dependability measures [6, 7]. However, in our case we have investigated the application-level behavior of networking programs under hardware faults.

VI. CONCLUSIONS

Faults in computer systems are causing system failures and silent errors. With the changing technologies, it is likely that the effects of faults will increase in the future. Although systems are built with increasingly complicated error detection and correction schemes, it is likely that such techniques will not be able to capture all faults. Therefore, there is an increasing need to measure the effects of faults of computer systems. In this paper, we have introduced metrics to measure errors in networking applications.

Particularly, we have selected seven representative applications and discussed how errors in these applications can be quantified. We then performed an analytical study, which investigated the effect of cache faults on these error measures. We plan to classify the faults into different according to their place of occurrence, i.e. during data processing or control logic execution of network applications. After classifying the faults an important task would be to control their occurrence in runtime.

ACKNOWLEDGMENT

We thank Seda O. Mekik for her invaluable input to this work. We also thank Rajarshi Mukherjee, David Nguyen, and Steve Lieberman for providing valuable feedback on this paper.

REFERENCES


Figure 7: Error Generation probability for URL application