Dynamic Error Mitigation in NoCs using Intelligent Prediction Techniques

Dominic DiTomaso†, Travis Boraten†, Avinash Kodi†, and Ahmed Louri‡,

†*Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701*
‡*Electrical and Computer Engineering, George Washington University, Washington, DC 20052*
dd292006@ohio.edu, tb206706@ohio.edu, kodi@ohio.edu, louri@gwu.edu

***Abstract*—Network-on-chips (NoCs) are quickly becoming the standard communication fabric for multi-core systems. As technology continues to scale down into the nanometer regime, device behavior will become increasingly unreliable due to a combination of aging, soft errors, aggressive transistor design, and process-voltage-temperature variations. Further, stringent timing constraints in NoCs are designed so that data can be pushed faster. The net result is an increase in errors which must be mitigated by the NoC. Typical techniques for handling faults are often reactive as they respond to faults after the error has occurred, making the recovery process inefficient in energy and time. In this paper, we take a different approach wherein we propose to use proactive, fault-tolerant schemes to be employed before the fault affects the system. We propose to utilize machine learning techniques to train a decision tree which can be used to predict faults efficiently in the network. Based on the prediction model, we dynamically mitigate these predicted faults through error correction codes (ECC) and relaxed timing transmission. Our results indicate that, on average, we can accurately predict timing errors 60.6% better than a static single error correction and double error detection (SECDED) technique resulting in an average 26.8% reduction in retransmitted packets, a average net speedup of 3.31×, and an average energy savings of 60.0% over other designs for real traffic patterns.**

I. Introduction

Today computer processors are made up of tens to hundreds of processing cores such as Intel's 48-core SCC [1], Tilera's 72-core TILE-Gx [2], and Kalray's 256-core MPPA [3]. Aggressive transistor scaling has accelerated the rapid growth in the number of cores that can be integrated on a single chip. In order for these cores to efficiently communicate, conventional bus-based networks have been replaced with network-on-chips (NoCs) [4], [5]. NoCs are scalable, modular designs that have emerged as the standard communication fabric in multicore systems. In NoCs, processing cores use routers connected by segmented links for efficient, scalable communication without global wire delays. However, the links between routers are a single point of failure in conventional NoC designs. Furthermore, as these links are pushed to their limits in terms of speed and technology, reliability often becomes a major challenge due to soft, or transient, errors.

The scaling of transistors has enabled the integration of billions of transistors on a chip. However, as these transistors continue to shrink, within-die variations in parameters such as process, voltage, and temperature are growing. Process variation impacts the design of different components by introducing slight variations due to the difficulty in controlling the fabrication process (such as optical proximity effects [6], [7], [8], dopant density fluctuations [9], etc.). Additionally, voltage variations can be caused by fluctuations in the power supply network. Finally, temperature variation can be caused by varying component utilization and other temporal and spatial variations. The VARIUS model [10] defines how these parameter variations can affect the transistors in such a way that some transistors become slower than others. Besides parameter variations, the switching of transistors can be further delayed by wear-out effects such as negative-bias temperature instability (NBTI) and hot carrier injection (HCI) [11], [12], [13], [14], [15].

Both parameter variations and wear-out effects can lead to data arriving slower than the expected time resulting in one type of soft errors called timing errors [16], [10]. On the other hand, soft errors that are independent of data arrival time are called data corruption errors in which single event upsets or crosstalk cause bits to unexpectedly change at any point in the data path [17]. Timing errors can occur in any stage of the NoC that has a timing constraint defined by the clock period. Typically, clock periods are set such that there is some slack between the mean data arrival time and the timing constraint. As power and performance are critical in NoCs, any increase in network frequency will improve performance [18], [19]; however this will put additional burden on the timing constraint. Further, data arrival times may become longer due to the insertion of global communication links [20], [19] which reduces hop count (to improve performance) or require less link repeaters (to save power). Therefore, with aggressive timing constraints and the occurrence of data corruption errors, fault mitigation techniques become critical.

Typical techniques for handling faults are often reactive which implies that they respond to faults after the error has already occurred. Reactive fault handling techniques are not the most optimized methods because they are employed after errors have already affected the performance of the system. Some reactive techniques route around faults after they have occurred [21], [22], [23], [24]; some use reconfigurable links

978-1-5090-3508-3/16/$31.00 © 2016 IEEE

to avoid faults [25], [26], [27]; while others detect all errors after they occur [28], [29]. On the other hand, proactive techniques prevent faults before they occur or reduce the probability of a fault affecting the packet. Proactive fault tolerant schemes can be more beneficial because they are employed before the error affects the system. Some proactive techniques include load balancing routers and links to prevent device wear-out [11], re-routing in dynamic voltage scaling systems to avoid low-voltage routers [16], and prediction of timing-critical instructions [30] or program phases [31].

Machine learning (ML) algorithms have been used in network/multi-core applications other than fault prediction for various applications such as network reconfiguration [32], [33], [34], detection of false memory sharing [35], management of voltage/frequncy/power network [36], [37], and prediction of congestion [38], [39]. Different ML models are implemented in these designs such as decision trees and artificial neural networks. Regardless of which model is selected, a data set is an important part of the model. The data set is typically separated into a training set and a testing set. After a model is processed on the training set, predictions are made against the test set to determine the performance of the model.

In this paper, we propose a comprehensive fault-prediction system in which we (a) create a methodology to obtain the training/testing sets, (b) train a ML algorithm to predict timing faults on links, and (c) mitigate soft errors. We focus on soft errors (timing and data corruption) during the link traversal stage. Since fault prediction with ML has never before been implemented for NoCs, we have developed data sets based on several effects such as device wear-out and process-voltage-temperature variation. From these data sets, we train a ML model which can predict timing faults during runtime and produce several different outcomes each time a flit uses a link: none of the bits will be in error, few bits will be in error (1-2 bits), or several bits will be in error ($>$ 2 bits). The model we use for predicting faults is a decision tree due to the low overhead during the testing phase which consists of a few comparisons instead of more complicated operations such as multiplication as seen with other ML models. Training for the ML algorithm can be done offline so that it does not affect the performance of the applications. To ensure correct execution of applications, most designs encode packets with a cyclic redundancy check (CRC) to detect errors. Based on the outcome of our predictor, we will decide whether to use only the baseline CRC or strengthen CRC. We can strengthen CRC by applying additional error correcting codes (ECC) or by applying relaxed timing transmission. The combined impact of prediction and mitigation is to reduce the retransmission of packets and save energy when soft errors manifest in the NoC.

The main component of our work is a proactive fault-tolerant scheme which uniquely predicts timing faults to improve our dynamic soft error mitigation technique. Mispredictions do not affect the correctness of execution; however they will only cause latency and energy consumption to increase. Our proactive scheme differs from other proactive techniques in that we propose a fine-grain prediction and mitigation of faults on a flit-by-flit basis rather than avoiding areas with high probability of faults. Unlike previous fault-tolerant designs, we use ML algorithms to precisely pinpoint faults through a statistical approach. Our main contributions include:

- **Design Data Sets:** We combine several effects such as device wear-out and process-voltage-temperature variation to determine the probability of two types of timing errors: a few bits in error and many bits in error. These probabilities are used to create data sets which are used to train and test our ML algorithm.
- **Fault Prediction:** Based on the developed data sets, we use ML techniques to train decision trees which are used to predict timing faults on the links. We analyze the effectiveness of our predictors on testing data sets.
- **Dynamic Error Mitigation:** Once the faults are predicted, we dynamically use ECC and relaxed timing transmission to mitigate soft errors as well as show the effects of our mitigation on network energy and performance.

Before we can train or test the ML algorithm, a suitable data set is required. A data set is a large list of samples consisting of a *label* and a set of *features*. The data set in this case indicates the number of errors (label) and the conditions under which the fault may have occurred (features). Therefore, our data set will, among other uses, provide a *true* label in which we can compare our *predicted* label. In Section II, we will describe how to obtain the features and true labels and Section III will describe the machine learning algorithm used to predict labels. Once we have our predicted label, Section IV describes the mitigation techniques we will use to avoid errors if necessary. Finally, we will evaluate the performance of our predictors as well as the effects of correct/incorrect predictions on network performance in Section V.

II. Design of Data Sets

To develop a data set, we first must create a fault model which can realistically produce a probability of error in a system based on a set of parameters. Using this fault model, we can then create a data set consisting of a large amount of samples by varying the model's input values. We can use this process to create different data sets for each link in the NoC. Figure 1 shows the complete methodology involved in obtaining a data set including the fault model shown in the dotted box. Subsection II-A explains our fault model which consists of several realistic temperature, delay, and variation models. Subsection II-B explains the methodology used to obtain our data sets.

A. Fault Model

Our fault model, shown in the dotted box of Figure 1, correlates link utilization to temperature and degree of wear-out to transistor delays. The corresponding temperature and delays are then passed to the VARIUS model [10] which incorporates process and voltage variations to determine the probability of timing errors.

The first input parameter for our model is link utilization. A high link utilization implies an increased number of link and

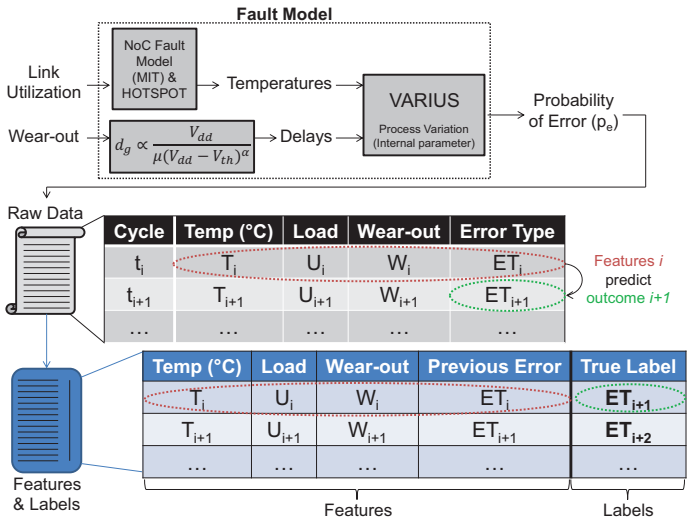


Fig. 1. Our process to create features and labels using fault models.

router traversals which lead to increased energy consumption and higher temperature. Therefore, the probability of fault will increase because gate delays become longer as temperature increases. Using a NoC fault model [17] which integrates the HotSpot thermal model [40], we correlate link utilization to temperature. We assume link utilization that ranges from 0.01 to 0.4 flits/cycle which corresponds to temperature values ranging between approximately 75-104 degree Celsius. In subsection II-B, we will explain why this range was selected and how the link utilization is varied.

The next input parameter of our model is device wear-out. Devices under long-term stress suffer from effects such as wear-out and failure due to NBTI and HCI [11]. NBTI and HCI shift transistor parameters over time and can be modeled through transistor threshold voltage (V_{th}) [41]. The shift in threshold voltage causes increased transistor delays according to the Alpha-power law [42]:

$$d_g \propto \frac{V_{dd}}{\mu(V_{dd}-V_{th})^{\alpha}} \quad (1)$$

where d_g is the transition delay, $\mu \propto T^{-1/5}$ (where T is temperature), and $\alpha = 1.3$. Therefore, switching based HCI stress and idle NBTI stress cause wear-out which increases the delays of transistors and devices become more susceptible to timing errors.

We consider a permanent fault in the device when ΔV_{th} is greater than 10% [41]. Therefore, we define three threshold ranges for wear-out which are below 10%: low (ΔV_{th}=0-3.3%), medium (ΔV_{th}=3.3-6.6%), and high (ΔV_{th}=6.6-10%). From Equation 1 we determine the corresponding transistor delays for each wear-out value. Assuming $V_{dd} = 1V$ and $V_{th0} = 0.15V$, the new delays (d) for each wear-out value based on the initial delays (d_0) are:

- **Low:** d = $1\times d_0$ to $1.00592\times d_0$
- **Medium:** d = $1.00592\times d_0$ to $1.01190\times d_0$
- **High:** d = $1.01190\times d_0$ to $1.01796\times d_0$

Based on the wear-out value, a range of delays is calculated and the average delay is input to the VARIUS model.

The final aspect of our fault model is the process variation which is integrated as an internal parameter of the VARIUS model. Process variation has both a systematic component which is spatially correlated and a random component which is not spatially correlated. The systematic variation is modeled by using a function which relates parameter correlation to distance. The relationship is negative and approximately linear. For example, two points very close together have transistor parameters which are highly correlated and two points very far away have very low correlation. At a distance of ϕ, two points are no longer correlated. We use $\phi = 1$ cm as experimental results show that gate lengths are correlated up to approximately half the chip length [6], [10]. Therefore, depending on link utilization, wear-out, and position on the chip, we can obtain a probability of error for each link.

B. Data Sets

Next, we must create raw data by (a) inserting various link utilization and wear-out values, and (b) determining what type of error occurred. In order to obtain a wide range of input values, we model an application which ramps up average link utilization from 0.01 to 0.4 flits/cycle, as shown in Table I. We stop at 0.4 flits/cycle as most networks saturate before this point and temperatures approach the bounds of normal operating temperatures [17]. This model application is run three separate times for each value of wear-out (low, medium, and high). The ML model is independent to the order in which the link utilization or wear-out is varied and only requires a wide range of values so that all different scenarios can be handled during runtime. First, to vary the link utilization, we start at a load of 0.01 flits/cycle and maintain this load for 3,000 cycles. At each cycle, a temperature is selected from a range of temperatures based on normal bounds for on-chip temperatures and is modeled as in [17]. For example, at a load of 0.01 flits/cycle, a temperature would be randomly selected between 75 and 77 degree Celsius. After 3,000 cycles, there is a ramp up period of 500 cycles in which the link utilization increases but the temperature slowly ramps up. This is to capture the delay in temperature increase after link utilization has been increased. Table I shows the change in link utilization and temperature for the model application. This model application allows our data set to have both a wide range of input values and a high number of samples.

The output of our fault model, as shown in Figure 1, is the probability of error (p_e) for a single bit line in each of the links. Our approach is generic and can be applied to many different scenarios such as various link widths, number of links, and network topologies. Each link has n data bit lines and there are a total of L links in the network. We consider a 64-core concentrated mesh topology with $n = 64$ and $L = 48$. Since all bit lines of a link are relatively close, it is assumed that p_e is the same for all n bit lines within a link. However, each individual wire of the L total links can have a different p_e.

TABLE I
MODEL APPLICATION LINK UTILIZATION AND TEMPERATURE PATTERN.

Load (flits/cycle)	Temperature (Celsius)	Duration (cycles)
0.01	75-77	3,000
0.1	76-80	500
0.1	78-82	3,000
0.2	80-89	500
0.2	87-93	3,000
0.3	90-99	500
0.3	95-101	3,000
0.4	97-101	500
0.4	98-104	3,000

Figure 1 shows two samples in our raw data. For each cycle t_i, a new sample is created which consists of the input temperature (T_i), utilization (U_i), and wear-out (W_i) as well as the error type (ET_i). To determine the error type, p_e is used to experimentally determine if an error occurred in each bit line. Since there are n bit lines in a link, this creates a bit error vector of size n, where the value of each index is 0 if there is no error and 1 if there is an error. From this bit vector we can determine the error type (ET) by counting the number of 1's in the bit vector: No error (0), Few Errors (1-2), or Many Errors (>2). A new sample is created for every cycle in each run of our model application.

From the raw data, we develop our data set which consists of features and true labels. Since we are interested in preventing errors before they occur, we must use currently available information (features) to predict errors in the future (labels). Therefore, when creating our data sets, we use inputs from cycle i as our features and outcomes from cycle $i+1$ as our true labels as shown in Figure 1. We can use the process described in this section to obtain a data set for each link so that we can predict errors on each link separately. Predicting errors separately for each link will improve overall accuracy of the predictions with a low overhead which will result in lower energy and better performance.

Finally, we randomly split the samples into training, testing, and validation data. The training data contains 60% of the samples whereas the testing and validation data each contain 20% of the remaining samples. Since the probability of error is typically very small, our data set is quite skewed. For example, if the probability of error is 1% then on average one sample out of every 100 will have an error. The skewed data set will make the decision tree training more difficult. Therefore, to remedy the skewed data, we use a common technique in which training samples are randomly replicated such that each label has an equal amount of samples (50,000 samples/label). Now each label will be fairly represented and machine learning algorithms can be used on the data as explained in the next section.

III. MACHINE LEARNING

The complete list of features we consider in our design are: Temperature (T), Utilization (U), Wear-out (W), and Previous Error Type (P). The feature values for Temperature are low

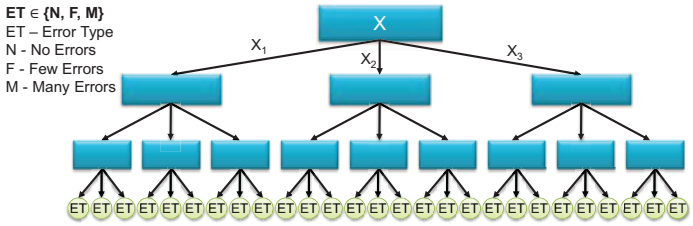


Fig. 2. Example of a generic decision tree with outcomes that determine the predicted error type.

(≤84 degree C), medium (≥85 degree C and ≤94 degree C), and high (≥95 degree C). The feature values for Utilization are low (≤0.1 flits/cycle), medium (>0.1 and <0.3 flits/cycle), and high (≥0.3 flits/cycle). The feature values for Wear-out are low, medium, and high as described in Section II-A. The feature values for Previous Error Type are "No Errors," "Few Errors," and "Many Errors."

Each feature can potentially have an effect on probability of error and we allow the machine learning algorithm to decide which are the more useful features for predicting errors. The features can be stored in a small table at each link. The values of some features, such as Temperature and Wear-out, change slowly over time; therefore, these features do not need frequent updating. Link temperature can be estimated by periodically gathering data from course-grained temperature sensors on the chip. The work in [11] correlates activity to wear-out. Therefore, link wear-out can be estimated by keeping a course-grained counter of link traversals. Link utilization provides more current information than link wear-out and can be used in making fine-grained decisions. Finally, Previous Error Type can easily be updated every time a prediction is made.

In our design, we use decision trees to predict errors on each of the links. Testing for decision trees consists of a small number of comparisons. The maximum number of comparisons equals the number of levels in the tree. In this paper, we choose the maximum size of the tree to be three levels as shown in the example tree in Figure 2. This allows our design to quickly predict a new outcome every cycle. Each node in the tree represents a feature (X) and the arrows represent feature values (X_i). For example, a feature could be "Temperature" and a feature value could be "low." The leaves of the tree represent the predicted outcome (ET) and can be any of the following: No Error (N), Few Errors (F), or Many Errors (M).

As each link is at a different location in the topology and process variation is spatially correlated, a separate decision tree is trained for each link. Training the decision tree is done offline and uses the ID3 algorithm [43]. In the ID3 algorithm, the criteria for creating nodes are information gains. Features with the largest information gain are selected as nodes in the tree. Algorithm 1 details the process of creating the decision tree based on the set of training data, D (previously described in Section II-B), and the set of features, F (Temperature, Utilization, Wear-out, and Previous Error Type). First, the ID3 algorithm is recursive with the terminating condition being all

Algorithm 1 ID3 algorithm used to build the decision trees [43].

```
ID3(Training data D, Feature F):
    if all samples in D have the same label:
        return a leaf node with that label
    let X∈F be the feature with the largest information gain
    let R be a tree root labeled with feature X
    let D_1, D_2, ..., D_k be the partition produced by splitting D on feature X
    for each D_i∈D_1, D_2, ..., D_k:
        let R_i=ID3(D_i, F-{X})
        add R_i as a new branch of R
    return R
```

examples in D have the same label. Next, the feature with the largest information gain is called X and R is the tree root labeled with feature X. Then, the initial data set D is partitioned into k separate data sets $(D_1, D_2, ..., D_k)$ each corresponding to a value of feature X. Finally, for each data set D_i, the ID3 algorithm is recursively called passing it the partitioned data set D_i and a set of features excluding feature X $(F - \{X\})$. The tree root which is returned by the recursive call (R_i) is added as a new branch to the original tree root R. After the ID3 algorithm is finished, a complete tree is built with nodes containing features that maximize the information gain on the training set. A modification we add to the ID3 algorithm is that we modify the terminating condition to limit the size of the tree to three levels so that the number of comparisons during testing is reduced to only three, resulting in small energy and latency.

Now we will detail the steps in building a decision tree in our design using an example training data set. Figure 3(a) shows the fully built decision tree and Figure 3(b) shows the steps in building this decision tree. The table titled D in Figure 3(b) is the example data set. There are nine samples in this data set and each sample has feature values. Some feature values are omitted by a dash (-) for illustrative purposes of the example. Also, each sample has a true label for the error type (ET).

Following the steps in Algorithm 1, it is not true that all samples in D have the same label so we can move on. Next, for this example, we assume Temperature has the largest information gain; Therefore, we let X = Temperature in Step 1 of Figure 3(b). Temperature is now the root of the decision tree. Next, in Step 2a, D is partitioned by splitting it on the feature values of Temperature (Low, Medium, and High). After the splitting, there are now three tables showing each data set, D_1, D_2, and D_3. Also, since Temperature was already used, we can remove it from the feature list in Step 2b.

Finally, Step 3 is to recursively call the ID3 algorithm for each newly partitioned data set. For example, the ID3 algorithm will be used on data set D_1 to find the node at the first branch of the tree. Since all of the samples in this partitioned data set have the label N, the label is returned and becomes a node in the tree. The ID3 algorithm will also be called on D_2 to find the node at the second branch in the tree. Since all of the samples do not have the same label and

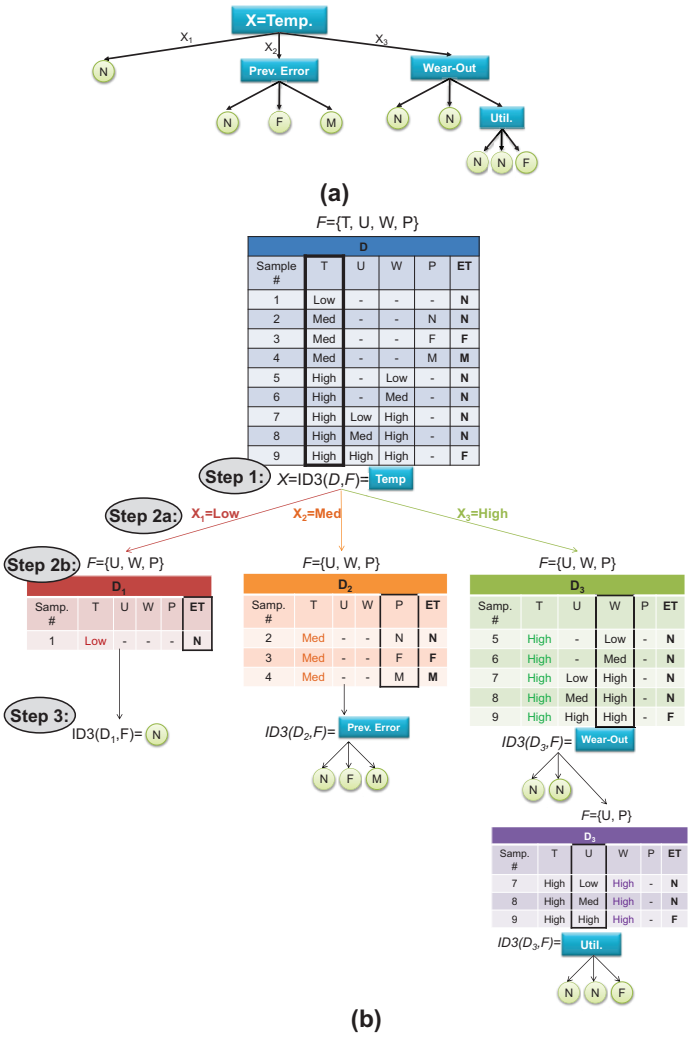


Fig. 3. (a) Example decision tree and (b) steps to build the tree. Step 1: Feature with the highest information gain becomes the root of the tree. Step 2a: Data set is partitioned on the feature values of the root. Step 2b: Root feature is removed from the feature list. Step 3: Recursively call ID3 on each newly partitioned data set. Repeat all steps until the tree is completed.

we assume Previous Error has the largest information gain, Previous Error becomes the next node in the tree. Steps 2a-3 are repeated: D_2 is split, the feature set is reduced further, and ID3 is called again which results in the labels N, F, and M. Finally, ID3 is called on D_3 which is the third branch corresponding to a high temperature. The steps are repeated until all samples have the same label or there are three levels in the tree. After all nodes in the tree have features or labels, the result is the completed tree in Figure 3(a).

IV. Error Mitigation and Router Microarchitecture

Once the errors can be predicted, the next step is mitigation of the errors. We add cyclic redundancy check (CRC) encoder blocks at the router ports coming from the cores and CRC decoders at the router ports going to the cores. The top of

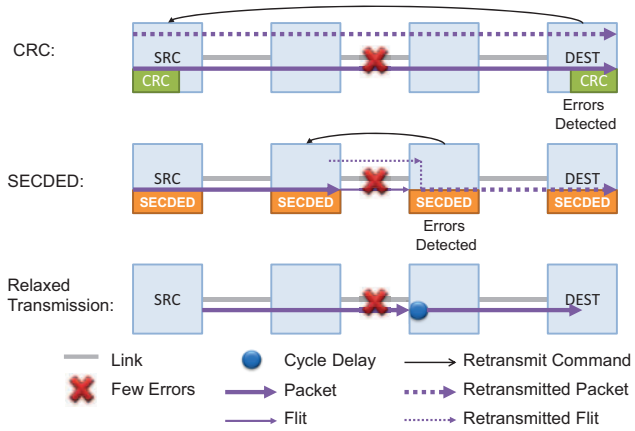


Fig. 4. Examples of CRC, SECDED, and relax transmission mitigation techniques.

Figure 4 shows an example of a packet transmission using CRC. When a packet is injected into the network, it is always encoded with CRC. Before the packet is ejected out of the network it is always decoded and checked for errors. If errors are detected at a packet's destination then the packet must be retransmitted from the source.

We use the IEEE standard CRC-32 [44]. CRC-32 is a commonly used error detection code capable of detecting all one to three bit errors, odd bit errors and a fraction of burst errors up to the size of the 32-bit check code. Each 32-bit check code is produced and appended to the tail of each packet as it is injected into the network, extending the packet size by 32-bits to a total of 256-bits. When an error is detected anywhere in the packet, we know the error exists but correction of the error is not possible with CRC decoding. Mitigation of the error is concluded with successful packet retransmission. CRC offers the most coverage in our network but has the highest overhead. Hence, CRC is reserved only for use at the source and destination routers. Since the source can often be several hops away from the destination, we will reduce the latency and energy overheads of retransmission by error mitigation at the intermediate hops between the source and destination.

At the intermediate hops, we use two forms of error mitigation: (a) hamming codes which have single error correction and double error detection (SECDED) and (b) relaxed transmission (RT). An example of packet transmission using SECDED is shown in the middle of Figure 4. We encode at the output port of each router with SECDED, then the flit traverses the link which is susceptible to faults, and finally we decode at the input port of the next router. If errors are detected and cannot be corrected then a retransmission must occur. However, only one flit must be retransmitted and the flit is only retransmitted one hop as opposed to being retransmitted from the source. If an error can be corrected then a retransmission can be avoided. For error correction, SECDED is implemented using a (72,64) hamming code with a maximum error correction tolerance of 1-bit per flit. All single bit errors in the 72-bit encoded flit will be corrected upon injection into the router. Overall, SECDED always has an energy overhead but only a latency overhead if a retransmit is required.

TABLE II
LATENCY OVERHEADS OF EACH MITIGATION TECHNIQUE GIVEN THE NUMBER OF TIMING ERRORS.

		CRC	SECDED	RT
True Outcome	No Errors	No Overhead	No Overhead	2 Cycle Delay
	Few Errors	Full Retransmission	No Overhead or 1 Hop Retrans.	2 Cycle Delay
	Many Errors	Full Retransmission	Full Retransmission	2 Cycle Delay

The second form of error mitigation is RT and is shown at the bottom of Figure 4. RT waits an additional cycle before reading the data at the input port; essentially, giving the flit twice as much time to traverse the link. This will relax the timing constraint on the transmission of the flit and reduce the probability of a timing error to very close to zero. RT can be done by, first, stalling the flit and sending a signal to the input demultiplexer of the downstream router. The downstream router is now notified to wait two cycles before reading data. After the router is notified, the flit begins the two cycle link traversal. Compared to a regular flit transmission, RT has no extra energy overhead but has an extra latency overhead of two cycles: one cycle to signal the downstream router and one additional cycle for data transmission. Overall, we can offer error mitigation on a hop-by-hop basis using SECDED or RT and we can offer greater error mitigation from source to destination using CRC.

Each error mitigation technique may have latency and energy overheads depending on different outcomes. Table II summarizes the latency overheads of each mitigation technique based on how many errors are there in the data, i.e. the true outcome. For CRC, if there are any errors, then the full packet will be fully retransmitted from the source. For SECDED, if there are no errors, then there will be no additional latency overhead. However, if there are a few errors, then there will be two possibilities: (1) a single error will be corrected and there will be no latency penalty, or (2) two errors will be detected in which only one flit will be retransmitted for one hop. If there are many errors, then it will only be detected by the CRC at the destination and a full retransmission will be required. For RT, there will be a two cycle delay for any number of timing errors.

Table III summarizes the energy overheads of each mitigation technique. For CRC, there will always be the energy to encode/decode as well as the possibility for additional energy due to a full retransmit. When SECDED is employed, there will be the SECDED encoder/decoder energy. Additionally, there can be an energy overhead from either a full retransmission or a one hop, one flit retransmission. The energy dissipation for a one hop retransmission will be much lesser than the energy dissipation for a full retransmission. Lastly, RT has no additional energy overheads for timing errors.

As each form of error mitigation has either a latency or energy overhead, we will reduce these additional overheads

TABLE III
ENERGY OVERHEADS OF EACH MITIGATION TECHNIQUE GIVEN THE NUMBER OF TIMING ERRORS.

		CRC	SECDED	RT
True Outcome	No Errors	CRC	SECDED	No Overhead
	Few Errors	CRC + Full Retransmission	SECDED and/or 1 Hop Retransmission	No Overhead
	Many Errors	CRC + Full Retransmission	SECDED + Full Retransmission	No Overhead

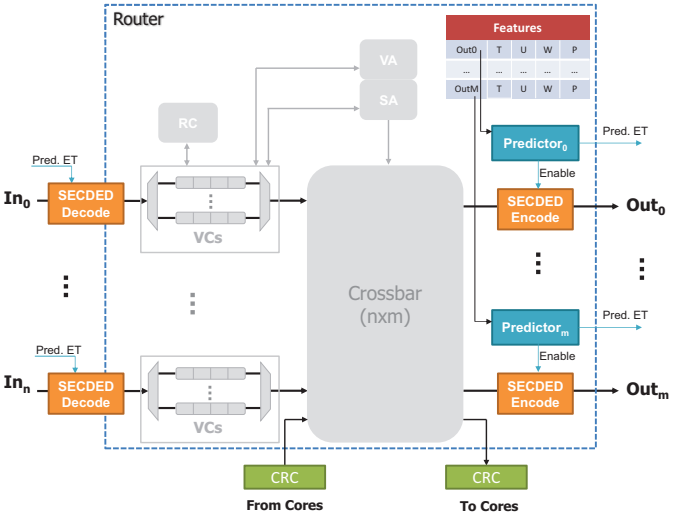


Fig. 5. Router microarchitecture showing the feature table, predictors, encoders, decoders, and CRC blocks.

by using prediction to dynamically enable and disable the SECDED and RT. Figure 5 shows the router microarchitecture in our network including the predictor blocks. There are also virtual channels (VCs) to buffer flits at the input ports and avoid deadlocks. We use a crossbar to switch the flits from the input ports to the output ports as well as output buffers to store packets at the output ports when needed. The router is a typical credit-based, pipelined router with route computation (RC), virtual channel allocation (VC), and switch allocation (SA) blocks. Routers can be connected to other routers via links in any topology. The CRC blocks are shown at the cores as we employ this error mitigation only at the source and destination. The SECDED encoders are at the output ports and the SECDED decoders are at the input ports as this error mitigation is used on a hop-by-hop basis.

Based on the output of the prediction algorithm, the network can choose when to enable SECDED, when to enable RT, or when to disable both. The features previously explained are stored in the features table shown in Figure 5. For each output port, the features table has an entry which stores the feature values for temperature (T), utilization (U), wear-out (W), and previous error type (P). Additionally, at each output port there is a predictor block ($predictor_0$ to $predictor_m$) which implements each link's unique decision tree using low overhead comparisons. The predictor takes a table entry as input and then outputs the predicted error type as well as an enable signal. Based on the predicted error type (ET) three different actions can occur: 1) If ET="No Errors" then no action is taken, 2) If ET="Few Errors" then the flit is encoded with SECDED before the link is used, and 3) If ET="Many Errors" then the flit is sent using RT. Therefore, if SECDED is used then the predictor enables the encoder and forwards the predicted ET to the downstream router so that it knows to decode the flit when it arrives. Similarly with RT, the predictor forwards the predicted ET to the downstream router so that the input port knows to wait an additional cycle before reading the data.

Figure 6 shows an example of packet transmission with and without prediction. With both prediction and no prediction, we assume CRC is always enabled in order to detect errors at the destination. Without prediction, SECDED and RT cannot be dynamically enabled and disabled in an intelligent manner. Therefore, in this particular example we assume SECDED is always enabled and RT is always disabled. With no prediction, energy is dissipated at every hop due to the SECDED encoders/decoders. If no errors occurred, then this energy dissipation was unnecessary. With prediction, when our design correctly predicts that no errors will occur, then the SECDED encoders/decoders can be disabled saving power. When a few errors are correctly predicted, the SECDED encoders/decoders are enabled and a few bit errors can be mitigated in the same manner as if SECDED is always enabled. Lastly, with no prediction, if many errors occur on the link, it will not be detected until the packet reaches the destination. However, with prediction, if many errors can correctly be predicted then RT can be employed possibly avoiding a full retransmission.

However, with any prediction technique there is also the possibility of misprediction. If our design incorrectly predicted that no errors occur then this misprediction will lead to a full retransmission. If our predictor incorrectly predicts that a few errors will occur then the SECDED will unnecessarily be activated, wasting energy. Lastly, a misprediction of many errors will lead to an unnecessary two cycle delay. Therefore, it is important that our predictor performs well so that the advantages of correctly predicting outweigh the misprediction penalties as evaluated in the next section.

V. EVALUATION

In this section, we first evaluate the overhead of the error mitigation and the predictor module. The energy and area of the links and other router components were evaluated with the DSENT NoC modeling tool [45] using a 45 nm technology library, a supply voltage of 1.0 V, and a frequency of 2 GHz. To evaluate our custom mitigation and predictor modules, the Synopsys Design Compiler with the same 45 nm technology library, 1.0 V supply voltage, and 2 GHz frequency was used.

Next, we discuss the results of the trained decision trees. Training is done offline; thus, there is no runtime latency for training. For each link in the network, we train a separate decision tree using separate training data sets. The decision trees are trained using the ID3 algorithm and each training data set initially has 50,000 samples. Since the data is skewed,

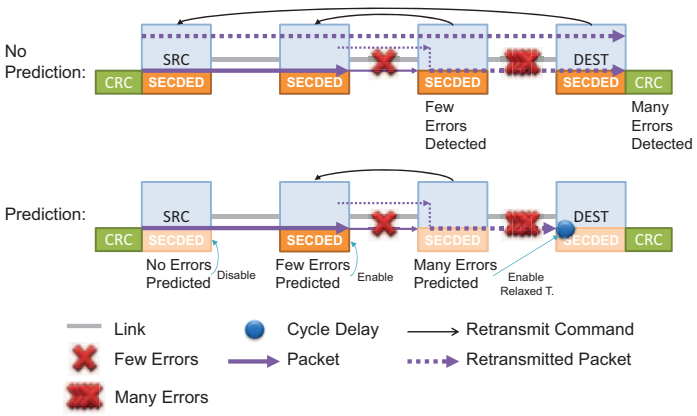
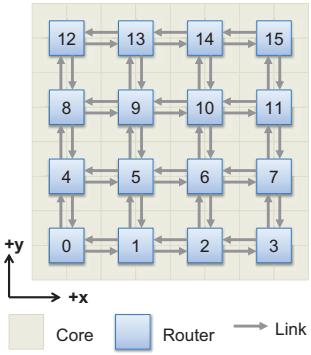


Fig. 6. Example of packet transmission with and without prediction.



Parameter	Value
Cores	64
Concentration	4 cores
Routers	16
Data Link Width	64 bits
Number of Links	48
Packet Size	4 flits
Flit Size	64 bits
VCs	4 per port
VC Size	4 flits

Fig. 7. Concentrated mesh with network parameters.

samples are replicated, as explained in Section II-B, to bring the total size of the training data to 150,000 samples. Then we test each decision tree using separate testing data sets for each link. Each testing data set has 10,200 samples. We compare our prediction design which labels samples using decision trees (DTs) to a technique which randomly assigns labels (Rand) and a technique which always applies SECDED.

Finally, we evaluate our design on a 64-core, concentrated mesh (CMesh) [46] network as shown in Figure 7. The network parameters are also shown in Figure 7. We use a packet size of 4 flits and each flit is 64 bits. We use ACK/NACK signals at the destination to signal successful/unsuccessful packet transmission and ACKs/NACKs per hop to signal successful/unsuccessful flit transmission [47]. Retransmission buffers are used at the routers which store the data until an ACK is received. If a NACK is received then the data is retransmitted. We assume that the ACKs/NACKs use control lines which are separate from the data lines. We inject timing errors on the data links every time a flit is transmitted. Additionally, to account for all other errors which are not timing related, we also inject data corruption faults which cause bits of data to be unexpectedly changed. Therefore, data corruption faults, unlike timing faults, are not due to late data arrival. We inject errors individually on each data link with a certain probability, based on our model described in Section II-A, which incorporates link utilization, temperature, process variation, and wear-out. For fair evaluation, the training and testing data follow the same probability distribution and the test data is independent of the training data per standards for all machine learning algorithms.

We execute real traffic traces on our network using workloads from the Splash-2, PARSEC, and SPEC CPU2006 benchmark suites. Traces were collected using the full execution-driven simulator SIMICS from Wind River [48] with the memory package GEMS [49]. The traffic collected is bimodal traffic with a mix of short (1 flit) packets and long (5 flit) packets. We assume a 2 cycle delay to access the L1 cache, a 4 cycle delay for the L2 cache, and a 160 cycle delay to access main memory. For each simulation, we warm up the simulator until a steady state is reached then we fix the number of packets sent (not including retransmissions) to 32,000 packets.

TABLE IV
OVERHEAD OF ROUTER COMPONENTS.

	Energy (pJ)	Area (μm^2)	Timing (ns)
64b Link	10.333	135.2	0.515
Buffer (1 flit)	1.154	1,017.3	0.07
Switch	1.572/flit	15,402 (8x8)	0.05

A. Overhead of Error Mitigation and Predictors

The overhead of the network components are shown in Table IV. The energy of a 64 bit, 4 mm long link was found to be approximately 10.333 pJ. In order to simulate a design with aggressive clocking, the delay of the link (0.515 ns) is designed to be 3% higher than the clock period (0.50 ns). Since the clock period is less than the average arrival time of the data, the link is susceptible to timing errors which we can mitigate with SECDED and RT.

Table V shows the energy, area, and timing overheads of the CRC, SECDED, and predictor modules. The energy for the CRC encoder and decoder are both 0.620 pJ. CRC uses several shift registers to obtain a check code. The value of this check code determines if there are errors or not. The process is the same at both the encoder and decoder; therefore, the energy, area, and timing are the same for both modules. The timing of the encoder/decoder is 1.09 ns. Since our clock period is 0.50 ns, encoding will take three cycles and decoding will also take three cycles. Overall the CRC accounts for approximately 18% of the total router energy and 14% of the total router area. The energy, area, and latency of CRC is the largest of our error mitigation techniques which is the reason why it is reserved only for source and destination routers.

For SECDED, the encoder energy is approximately 0.072 pJ which is approximately 2.6% of the total router energy. The decoder energy is the same if there are no errors, but higher if errors are corrected or detected. Since our SECDED uses a (72,64) hamming code, there is also an energy overhead for transmitting the additional 8 parity bits across the network links. The energy for parity bit transmission is approximately

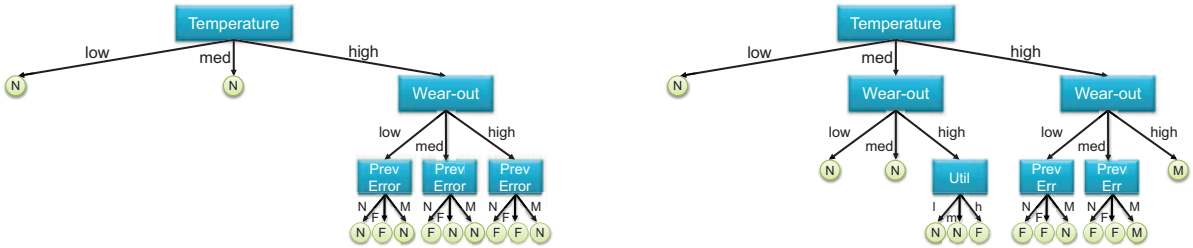


Fig. 8. Decision trees built using the ID3 algorithm for a +x link and a +y link at router 0.

TABLE V
ERROR MITIGATION AND PREDICTOR OVERHEADS.

Module	Energy (pJ)	Area (μm^2)	Timing (ns)
CRC Encoder	0.620	2,680.0	1.09
CRC Decoder	0.620	2,680.0	1.09
CRC Total	**1.240**	**5,360.0**	**2.18**
SECDED Encoder	0.072	285.6	0.24
SECDED Decoder	0.227	930.5	0.38
SECDED Parity	1.292	-	-
SECDED Total	**1.591**	**1,216.1**	**0.62**
Predictor Total	**0.002**	**29.8**	**0.17**

1.292 pJ which is 11% of the total link energy. The latency for SECDED encoding is only 0.24 ns and can be combined with the output buffering router stage. Using the Synopsys Design Compiler, the latency of the buffer stage is only 0.07 ns. Therefore, buffering and SECDED encoding will take a total of 0.31 ns which is well below our clock period of 0.50 ns. Similarly, SECDED decoding can be combined with the buffer write stage at the input port.

Finally, our predictor module consumes a very low energy of 0.002 pJ and occupies only 29.8 μm^2 of area. The predictor module implements the decision tree which uses, at most, three comparisons for each level in the tree; therefore, the overhead is very small. The latency of the predictor module is only 0.17 ns which is under the clock period, so predictions can easily be made every cycle.

B. Trained Decision Trees

Using the training data sets and the ID3 algorithm we chose to train a separate decision tree for each link in the topology due to the spatial correlation of process variation. While we chose CMesh, our approach is generic and can be applied to any topology. Figure 8 shows the resulting decision trees for a couple different links. Temperature is the root node for each tree in Figure 8 and was found to be the root node for every link in the topology. As temperature has such a strong effect on probability of timing errors, this is expected. At low temperatures, the probability of error is low. Therefore, 45 of the 48 trees (93.8%) have the N label as the leaf of the low temperature branch, the other trees have at least one F label in the subtree of the low temperature branch. No tree has a M label anywhere in the low temperature branch. At the second level, 46 of the decision trees had the Wear-out feature or a label at each node. The remaining two trees had a combination of Utilization and Wear-out at the second level. The Wear-out feature was so common because it is another strong predictor of timing errors. At the third level of the trees, the nodes vary mostly between the Utilization and Previous Error features with only two trees using the Wear-out feature. The Previous Error feature, being a weaker predictor at the bottom of the tree, has a high entropy, or uncertainty, which results in leaves that have less correlation to the feature values. However, the inclusion of multiple features in the decision trees results in more accurate predictions.

C. Performance of Decision Trees on Timing Errors

Next, we will examine the confusion matrix for our decision trees. A confusion matrix is one way to show the performance of a prediction algorithm. Each column in the matrix is a predicted label and each row is the true label. The confusion matrix shows how often a predictor confuses labels. The confusion matrix for the +x link of router 0 is shown in Table VI. The diagonals of this matrix show the number of test samples that were correctly labeled. For example, 7,508 out of the 10,200 test samples were correctly labeled N. The non-diagonal elements of the matrix show the number of test samples that were mislabeled. For example, 2,552 samples were labeled F when the true label was actually N.

The average confusion matrix for all the decision trees is shown in Table VII. When the sample is correctly labeled, there are no additional overheads. However, mislabeled samples will cause either additional latency or energy overheads. Table II and Table III from Section IV detail the overheads for each possibility in the confusion matrix. For example, if a sample is labeled F when the true label was actually N then the flit will be encoded/decoded with SECDED. However, since there were actually no errors, there will be a slight energy overhead due to the unnecessary SECDED encoding/decoding but no latency overhead. The exact energy of SECDED and CRC encoding and decoding was shown in Table V and the effects of the mislabeled samples on network performance will be shown in Section V-D.

From the average confusion matrix, we calculate the accuracy of our decision trees (DTs) compared to Rand and SECDED. The results are shown in the first column of Table VIII. The DTs correctly predict the true label in approximately 70.6% of the samples which improves over a random labeling

TABLE VI
CONFUSION MATRIX FOR +X LINK OF ROUTER 0.

		Predicted		
		N	F	M
True	N	7,508	2,552	0
	F	22	118	0
	M	0	0	0

TABLE VII
CONFUSION MATRIX AVERAGED OVER ALL DECISION TREES.

		Predicted		
		N	F	M
True	N	5,981.8	1,902.8	483.1
	F	121.1	565.0	334.1
	M	0.8	159.8	651.5

and SECDED which always assumes that a few errors will occur. Since the probability of error is low, many samples will have no errors which is called a skewed data set. Therefore, a common and better metric to consider is called the F-score. The F-score is a different measure of accuracy used for skewed data and is an average of the precision and recall which are measures of relevance. Table VIII shows the average F-score for each labeling technique. The DTs have the highest F-score at 61.0%. On average, DTs have a F-score 41.3% higher than Rand and SECDED.

Some mispredictions cause a small one cycle delay or a small encoding overhead. However, there are three cases in the confusion matrix in which mislabeling in our design will cause a full retransmission. These cases are in the lower left triangular part of the confusion matrix: 1) Predicted N/True F, 2) Predicted N/True M, and 3) Predicted F/True M. As these mispredictions are most costly, the total number of these three cases should be minimized. The third column of Table VIII shows the percentage of samples which will require retransmissions due to a timing error. In our design, on average, only 2.8% of the samples cause a full retransmission compared to 8.6% and 8.0% for Rand and SECDED, respectively.

D. Network Performance

After evaluating the performance of the decision trees on timing errors, we now introduce data corruption errors and evaluate the effect of all soft errors on network performance. First, in Figure 9, we show the percent of retransmitted packets. The percent of retransmitted packets for each design is relatively high due to the aggressive clocking, the multi-hop communication, and the high operating temperatures we assume. However, DTs can reduce the number of transmissions by 26.8% on average over the other designs. SECDED can prevent retransmission due to a few timing errors or a few data corruption errors; however, with many errors, a full retransmission is still required. The Rand labeling incurs the most retransmissions due to the high number of mispredictions.

In Figure 10, we evaluate application speedup relative to Rand which is equal to the ratio of execution time of a given design to the execution time of Rand. Averaged over all applications, DTs execute faster than other designs with a speedup of approximately 3.31×. The ability of DTs to accurately predict timing errors reduces latency delays such as one-hop retransmissions, full retransmissions, and unnecessary RT activations. Also, the ability of SECDED to detect and correct data corruption errors reduces transmissions; thereby, reducing the execution times. The static SECDED design always applies SECDED so it can avoid some one-hop flit retransmission; however, many errors will cause a full retransmission. The Rand design has a long execution time on average due to the large number of mispredictions.

TABLE VIII
ACCURACY, F-SCORE, AND PER-HOP RETRANSMIT PERCENT DUE TO TIMING ERRORS FOR OUR DECISION TREES (DTS) COMPARED TO OTHER LABELING TECHNIQUES.

	Accuracy	Fscore	Retransmit
DT	70.6%	61.0%	2.8%
Rand	33.3%	33.3%	8.6%
SECDED	10.0%	6.1%	8.0%

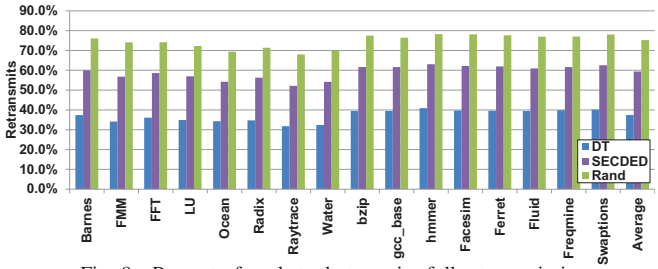


Fig. 9. Percent of packets that require full retransmission.

Finally, we examine the component breakdown for the energy per flit in Figure 11. Due to space constraints, we only show nine applications. During simulation, we calculate the total energy consumption for all the flits then we divide by the number of error-free flits which is fixed to 128,000 flits, or 32,000 packets, for all simulations. Therefore, designs with a higher number of retransmissions will require more energy to successfully transmit 128,000 error-free flits and will have higher energy per flit values. Our DT design reduces energy per flit by approximately 60.0% on average over all other designs. The main reason for this reduction is the fewer number of one-hop and full retransmissions due to correct predictions. Additionally, DTs can reduce the SECDED energy over SECDED and Rand when predictions are correct.

E. Other Failure Rates

Our design targets aggressive timing constraints in which the mean data arrival time is 3% longer than the clocking period. In this section, we will evaluate the effects of more

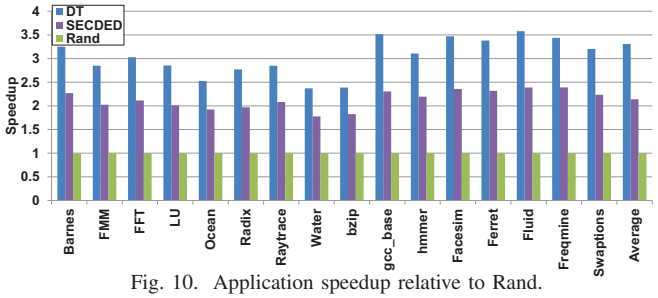


Fig. 10. Application speedup relative to Rand.

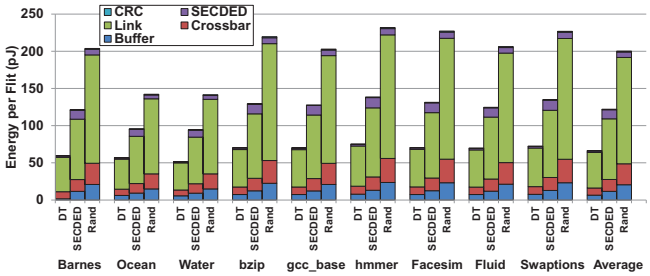


Fig. 11. Breakdown of energy per flit for different design.

TABLE IX
ACCURACY, FSCORE, AND PERCENT RETRANSMITS FOR DESIGNS WITH MEAN DATA ARRIVAL TIMES 1% LONGER (+1%) AND 1% SHORTER (-1%) THAN THE CLOCKING PERIOD.

+1%	Accuracy	Fscore	Retrans.	-1%	Accuracy	Fscore	Retrans.
DT	71.1%	51.8%	1.0%	DT	77.7%	46.1%	0.1%
Rand	33.4%	33.3%	2.9%	Rand	33.4%	33.6%	0.5%
SECDED	5.6%	3.5%	1.5%	SECDED	1.5%	1.0%	0.0%

relaxed timing constraints which will effectively lower the fail rates of the links. Table IX shows the accuracy and retransmit percentage of a mean arrival time 1% longer than the clocking period (+1%) and the accuracy of a mean arrival time 1% shorter than the clocking period (-1%). Our DTs have been trained and tested with the new mean arrival times. Due to the lower failure rates, DTs have less of an advantage as the timing constraint becomes more relaxed. However, as shown in Figure 12, DTs can still reduce energy consumption due to dynamic SECDED enabling and less retransmissions.

VI. CONCLUSIONS

With NoCs becoming the communication standard for multi-core systems, the reliability of network components becomes an important concern. Network links, in particular, are susceptible to timing errors due to stringent timing constraints, parameter variation, and device wear-out. Proactive fault-tolerant techniques can prevent errors before they occur or reduce the probability of faults. In this paper, we develop a new approach to proactive fault-tolerance which uses machine learning (ML) algorithms to predict and mitigate errors.

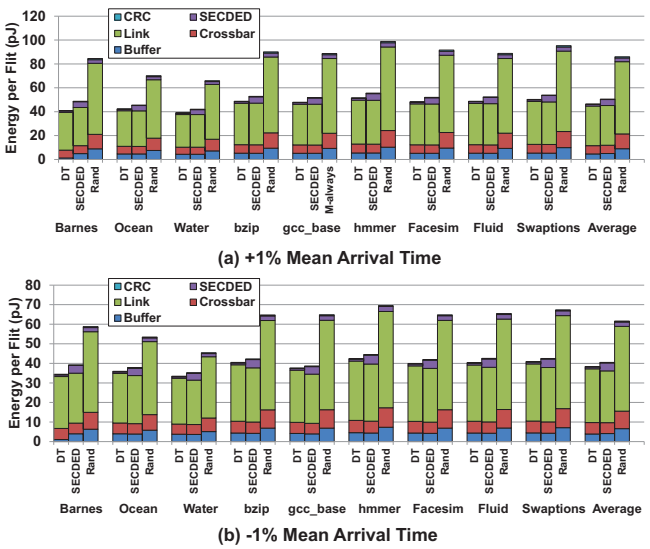


Fig. 12. Energy per flit breakdown for (a) a +1% mean arrival time and (b) a -1% mean arrival time relative to the clock period.

We provide a comprehensive fault-prediction system in which we (a) create a methodology to obtain realistic data sets, (b) train a ML algorithm to predict timing faults on links, and (c) mitigate for soft errors. We develop a fault model, which accounts for parameter variation and device wear-out, to create training/testing data sets for the ML algorithm. Using the training data set and the ID3 algorithm, we create decision trees which can be used to accurately predict the number of errors. Finally, we dynamically mitigate the errors using a combination of error correction codes (ECC) and a relaxed transmission. Our results show that the energy (0.002 pJ), area (29.8 μm^2), and latency (0.17 ns) overheads of the predictor implementation are minimal. Decision trees are able to accurately predict timing errors 60.6% better than a static SECDED technique. Our network results indicate a 26.8% reduction in packet retransmissions, a 3.31$\times$ speedup, and an energy savings of 60.0% on average over other designs.

ACKNOWLEDGMENT

This research was partially supported by NSF grants CCF-1054339 (CAREER), CCF-1420718, CCF-1318981, ECCS-1342657, CCF-1513606, CCF-1547034, CCF-1547035, CCF-1565273, and CCF-1600820. We would like to thank the anonymous reviewers for their valuable feedback.

REFERENCES

[1] J. Held, "Single-chip cloud computer: An experimental many-core processor from intel labs." Presented at Intel Labs Single-chip Cloud Computer Symposium, Santa Clara, California, 2010.

[2] M. Mattina, "Architecture and performance of the tile-gx processor family," *White Paper, Tilera Corporation*.

[3] B. de Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P. de Massas, F. Jacquet, S. Jones, N. Chaisemartin, F. Riss, and T. Strudel, "A clustered manycore processor architecture for embedded and accelerated applications," in *High Performance Extreme Computing Conference (HPEC)*, 2013.

[4] L. Benini and G. D. Micheli, "Networks on chips: A new soc paradigm," *IEEE Computer*, vol. 35, pp. 70–78, 2002.

[5] W. J. Dally and B. Towles, "Route packets, not wires," in *Proceedings of the Design Automation Conference (DAC)*, Las Vegas, NV, USA, June 18-22 2001.

[6] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos, "Modeling within-die spatial correlation effects for process-design co-optimization," in *Sixth International Symposium on Quality of Electronic Design*, 2005.

[7] M. Orshansky, L. Milor, and C. Hu, "Characterization of spatial intrafield gate cd variability, its impact on circuit performance, and spatial mask-level correction," *IEEE Transactions on Semiconductor Manufacturing*, vol. 17, no. 1, pp. 2–11, Feb 2004.

[8] B. Stine, D. Boning, and J. Chung, "Analysis and decomposition of spatial variation in integrated circuit processes and devices," *IEEE Transactions on Semiconductor Manufacturing*, vol. 10, no. 1, pp. 24–41, Feb 1997.

[9] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *Design Automation Conference*, 2004.

[10] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, Feb 2008.

[11] H. Kim, A. Vitkovskiy, P. V. Gratz, and V. Soteriou, "Use it or lose it: Wear-out and lifetime in future chip multiprocessors," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[12] JEDEC Solid State Technology Association, "Failure mechanisms and models for semiconductor devices," JEP122G, 2011.

[13] F. Oboril and M. Tahoori, "Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level," in *International Conference on Dependable Systems and Networks (DSN)*, 2012.

[14] D. Ancajas, J. Nickerson, K. Chakraborty, and S. Roy, "Hci-tolerant noc router microarchitecture," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013.

[15] K. Bhardwaj, K. Chakraborty, and S. Roy, "Towards graceful aging degradation in nocs through an adaptive routing algorithm," in *49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2012.

[16] A. Ansari, A. Mishra, J. Xu, and J. Torrellas, "Tangle: Route-oriented dynamic voltage minimization for variation-afflicted, energy-efficient on-chip networks," in *20th International Symposium on High Performance Computer Architecture*, 2014.

[17] K. Aisopos, C.-H. Chen, and L.-S. Peh, "Enabling system-level modeling of variation-induced faults in networks-on-chips," in *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2011, pp. 930–935.

[18] A. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. Das, "A case for dynamic frequency tuning in on-chip networks," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[19] C.-H. O. Chen, S. Park, T. Krishna, S. Subramanian, A. P. Chandrakasan, and L.-S. Peh, "Smart: A single-cycle reconfigurable noc for soc applications," in *Automation Test in Europe Conference Exhibition Design (DATE)*, 2013, pp. 338–343.

[20] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: Cost-efficient topology for high-radix networks," in *Proceedings of 34th Annual International Symposium on Computer Architecture(ISCA)*, June 2007, pp. 126 – 137.

[21] A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011, pp. 298–309.

[22] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A highly resilient routing algorithm for fault-tolerant nocs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 21–26.

[23] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," *SIGARCH Comput. Archit. News*, vol. 32, no. 2, 2004.

[24] J. Kim, C. A. Nicopoulos, D. Park, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, Boston, MA, USA, June 17-21 2006, pp. 4–15.

[25] W.-C. Tsai, D.-Y. Zheng, S.-J. Chen, and Y.-H. Hu, "A fault-tolerant noc scheme using bidirectional channel," in *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011.

[26] D. DiTomaso, A. Kodi, and A. Louri, "QORE: A fault tolerant network-on-chip architecture with power-efficient quad-function channel (QFC) buffers," in *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[27] R. Parikh, R. Das, and V. Bertacco, "Power-aware nocs through routing and topology reconfiguration," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[28] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures," in *to appear in The 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[29] R. Parikh and V. Bertacco, "Formally enhanced runtime verification to ensure noc functional correctness," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[30] J. Xin and R. Joseph, "Identifying and predicting timing-critical instructions to boost timing speculation," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[31] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke, "Trace based phase prediction for tightly-coupled heterogeneous cores," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 445–456.

[32] Y. Wang, M. Martonosi, and L.-S. Peh, "A supervised learning approach for routing optimizations in wireless sensor networks," in *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality*, 2006.

[33] J. S. Shen, C. H. Huang, and P. A. Hsiung, "Learning-based adaptation to applications and environments in a reconfigurable network-on-chip," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2010, pp. 381–386.

[34] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty, "Optimizing 3d noc design for energy efficiency: A machine learning approach," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '15, 2015, pp. 705–712.

[35] S. Jayasena, S. Amarasinghe, A. Abeyweera, G. Amarasinghe, H. De Silva, S. Rathnayake, X. Meng, and Y. Liu, "Detection of false sharing using machine learning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013.

[36] J. Won, X. Chen, P. V. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management," in *The 20th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[37] D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '13, 2013, pp. 8:1–8:10.

[38] E. Kakoulli, V. Soteriou, and T. Theocharides, "Intelligent hotspot prediction for network-on-chip-based multicore systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 418–431, 2012.

[39] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Q-learning based congestion-aware routing algorithm for on-chip network," in *IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications (NESEA)*, Dec 2011, pp. 1–7.

[40] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[41] Y. Wang, S. Cotofana, and L. Fang, "A unified aging model of nbti and hci degradation towards lifetime reliability management for nanoscale mosfet circuits," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, June 2011.

[42] T. Sakurai and A. Newton, "Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, Apr 1990.

[43] J. R. Quinlan, "Induction of decision trees," *MACH. LEARN*, vol. 1, pp. 81–106, 1986.

[44] K. Brayer and J. J. L. Hammond, "Evaluation of error detection polynomial performance on the autovon channel," in *National Telecommunications Conference*, December 1975, pp. 8–21.

[45] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, May 2012.

[46] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)*, Cairns, Australia, June 28-30 2006, pp. 187–198.

[47] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. Das, "Exploring fault-tolerant network-on-chip architectures," in *International Conference on Dependable Systems and Networks*, 2006.

[48] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hllberg, J. Hgberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, pp. 50–58, 2002.

[49] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's genreal execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, no. 4, pp. 92–99, November 2005.