High-performance, Energy-efficient, Fault-tolerant Network-on-Chip Design Using Reinforcement Learning

Ke Wang*, Ahmed Louri*, Avinash Karanth†, Razvan Bunescu†
*Department of Electrical and Computer Engineering, George Washington University, Washington, DC, USA
†School of Electrical Engineering and Computer Science, Ohio University, Athens, Ohio, USA
Emails: cory@gwu.edu, louri@gwu.edu, karanth@ohio.edu, bunescu@ohio.edu

***Abstract*—Network-on-Chips (NoCs) are becoming the standard communication fabric for multi-core and system on a chip (SoC) architectures. As technology continues to scale, transistors and wires on the chip are becoming increasingly vulnerable to various fault mechanisms, especially timing errors, resulting in exacerbation of energy efficiency and performance for NoCs. Typical techniques for handling timing errors are reactive in nature, responding to the faults after their occurrence. They rely on error detection/correction techniques which have resulted in excessive power consumption and degraded performance, since the error detection/correction hardware is constantly enabled. On the other hand, indiscriminately disabling error handling hardware can induce more errors and intrusive retransmission traffic. Therefore, the challenge is to balance the trade-offs among error rate, packet retransmission, performance, and energy. In this paper, we propose a proactive fault-tolerant mechanism to optimize energy efficiency and performance with reinforcement learning (RL). First, we propose a new proactive error handling technique comprised of a dynamic scheme for enabling per-router error detection/correction hardware and an effective retransmission mechanism. Second, we propose the use of RL to train the dynamic control policy with the goals of providing increased fault-tolerance, reduced power consumption and improved performance as compared to conventional techniques. Our evaluation indicates that, on average, end-to-end packet latency is lowered by 55%, energy efficiency is improved by 64%, and retransmission caused by faults is reduced by 48% over the reactive error correction techniques.**

I. Introduction

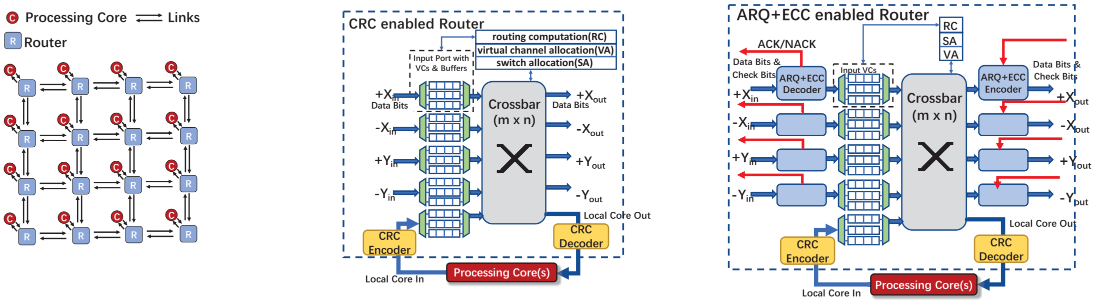
At the chip level, Network-on-Chips (NoCs) [1] are the standard interconnect solution. With continuing aggressive technology scaling, transistors and wires on the chip are becoming more vulnerable to permanent defects (hard faults) and transient faults (soft errors). While hard faults and soft errors are equally important for NoCs, this paper only concentrates on soft errors, in particular timing errors. The effects of these faults on the NoC communication substrate are immense, as they can lead to network disruptions, misrouting, protocol-level deadlocks, corrupted packets, incoherent caches, and erroneous memory traffic - all of which can lead to increased execution time, excessive delays, and increased power consumption while recovering from the fault [2]–[5].

A number of fault-tolerant techniques [2]–[5] have been proposed to enhance NoC's reliability. These methods include both reactive techniques that passively respond to faults after they occur [2] and proactive techniques that predict the occurrence of faults and mitigate those faults by deploying either load balancing [3] or deflection routing [4] techniques. Reactive techniques rely on forward error detection/correction and packet retransmission which can lead to excessive power consumption, extra delays, additional hardware and chip area. Proactive techniques, on the other hand, need to implement rules and strategies for prediction which can also occupy chip area and consume power. However, by predicting error and retransmitting packets before being requested, proactive techniques have the potential to improve performance and energy-efficiency. Unfortunately, manually designing the rules and strategies for making proactive predictions and decisions in NoCs requires substantial cognitive and engineering efforts.

Machine Learning algorithms, however, can train proactive techniques using complex data from various computer hardware designs [6], [7]. Previous research [6] deployed supervised learning with decision trees (DT) to predict the timing error rate of each link associated with the router and mitigate the predicted faults. However, supervised learning requires labeled training examples, furthermore the design of the control policy in prior approaches with supervised learning still requires human engineering. In this paper, we propose to use reinforcement learning (RL), which eschews the prediction step and automatically learns a decision policy that directly maps runtime NoC states to optimal decisions [8], [9].

First, we propose a new proactive fault-tolerant scheme which improves performance and energy efficiency for NoCs. The proposed fault-tolerant scheme allows routers to switch among four different fault-tolerant operations. Each operation mode has different trade-offs among fault-tolerant capability, retransmission traffic, latency, and energy efficiency. Second, we propose an RL-based control policy for the proposed fault-tolerant scheme. Per-router RL agents independently observe a set of NoC system parameters at runtime, and over time they evolve optimal per-router control policies. By automatically and optimally switching among the four fault-tolerant modes, the trained control policy results in minimizing system-level network latency and maximizing energy efficiency while detecting and correcting soft errors. Third, we present our simulation and evaluation studies using real-world benchmarks. The proposed RL-based fault-tolerant scheme prevents

978-3-9819263-2-3/DATE19/©2019 EDAA



(a) Typical 2D Mesh NoC

(b) CRC enabled router

(c) ARQ+ECC enabled router

Fig. 1. NoC and Router Architecture. (a) 4 × 4 2D Mesh NoC. (b) Cyclic redundancy check (CRC) enabled router.(c) Automatic retransmission query (ARQ) with error correction codes (ECC) enabled Router.

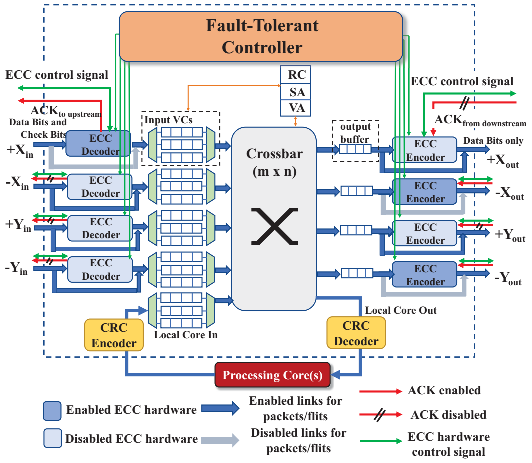


Fig. 2. Micro-architecture of proposed fault-tolerant router.

48% of unnecessary retransmission traffic. Simultaneously, the proposed methodology results in an average of 55% reduction in end-to-end packet latency, 64% more energy efficiency, and 46% reduction in dynamic power consumption over reactive fault-tolerant mechanism. The proposed methodology also outperforms conventional supervised learning-based technique using decision trees [6]: the proposed design has 10% lower end-to-end latency, 15% greater energy efficiency, and 17% reduction in dynamic power consumption.

II. Background

A typical NoC router in a mesh topology is comprised of five input ports and five output ports, as shown in Fig. 1(a). Data is transmitted in the form of *packets* which are segmented into several *flits* in NoC for efficient router resource utilization.

Since transient faults may manifest during any stage of transmission, a number of fault-tolerant techniques have been published. One basic soft error tolerant technique is to apply error detection at the local core input/output ports of the routers using Cyclic Redundancy Check (CRC), shown in Fig. 1(b). CRC is an error checking algorithm to detect accidental changes to raw data [10]. In a CRC-enabled NoC system, each flit of a packet is encoded with a CRC encoder before its injection into the *source* router. The flits will be decoded by a CRC decoder of the *destination* router. If the flit fails to pass CRC, a retransmission request will be sent to the source router, and the entire packet will be retransmitted from source. Such retransmission scheme is inefficient since a fault can only be detected at the destination.

To strengthen CRC with more powerful fault-tolerant capability, the Automatic Retransmission Query (ARQ) [11] protocol with error correction codes (ECC) is applied to every router to enable error detection in intermediate routers as shown in Fig. 1(c). ARQ+ECC is an error detection and retransmission method that relies on sending error checking result messages, in the form of *acknowledgement (ACK) flits*, between any two adjacent routers. Comparing to the CRC-based router, an additional ARQ+ECC encoder is assigned to each output port, while ECC decoders are added to input ports. ARQ+ECC adds redundant bits to flits to provide the function of single-bit error correction and double-bit error detection (SECDED) by the receiver. When a flit is transmitted, a copy of that flit is buffered in current router's virtual channel (VC) until it receives an ACK message back from the downstream router. If a negative-acknowledgement (NACK) is received, the buffered flit will be retransmitted to the downstream router.

III. Proposed Fault-Tolerant NoC Design

Fig. 2 illustrates the proposed fault-tolerant router which is capable of dynamically enabling/disabling ARQ+ECC hardware. As compared to the conventional fault-tolerant router design shown in Fig. 1(c), a per-router fault-tolerant controller along with output flit buffers are added. For the ease of explanation, we intend to introduce the following concepts with simplified router diagram in Fig. 3, wherein only ECC units are shown: 1) the channel from router_i to router_i+1 is defined as *channel_i*; 2)*ECC-Link_i* consists of the ARQ+ECC encoder of router_i and the ARQ+ECC decoder of router_i+1; 3) Enabling/disabling an ECC-Link_i means both the ECC encoder of router_i and the ECC decoder of router_i+1 are switched on, or powered off and bypassed, controlled by the fault-tolerant controller of router_i.

The fault-tolerant router dynamically deploys one of four proposed fault-tolerant operation modes, assigned for four different situations with various error levels on each link by the associated router, for optimal performance. The operation modes are described below:

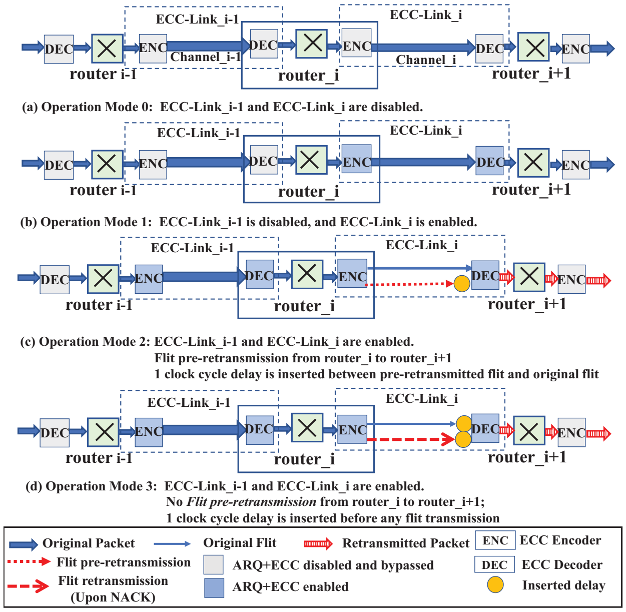
- **Operation Mode 0 - Minimum error level (Fig. 3(a)):** Under this scenario, there is a minimum possibility that transmitted flits would contain faults. Therefore, the overhead of deploying channel_i ECC hardware is larger than the overhead of possible packet retransmission when ECC hardware is disabled. In this case, ECC-Link_i and ECC-Link_i-1 are disabled to save power and reduce latency.

- **Operation Mode 1 - Low error level (Fig. 3(b)):** In this case, the overhead of enabling channel_i ECC hardware is smaller than the packet retransmission when ECC hardware is disabled. Nevertheless, most of the faults will be corrected by SECDED of ECC. In this mode, router_i enables ECC-Link_i to avoid full packet retransmission and consequently reduces power and latency.
- **Operation Mode 2 - Medium error level (Fig. 3(c)):** This is the situation wherein the faults on channel_i will not be corrected by downstream ARQ+ECC, and a flit retransmission from router_i is required. In this case, router_i anticipates a NACK message from router_i+1 requiring retransmission of the flit. Instead of waiting for the NACK, router_i pre-transmits the flit ahead of time to save the round trip flit transmission time in channel_i. We call this strategy *flit pre-retransmission*. To differentiate the transmission and pre-retransmission of the flit, we insert a one-clock-cycle delay between the two.
- **Operation Mode 3 - High error level (Fig. 3(d)):** This is the situation where the errors in transmitted flits will not be corrected by downstream ECC hardware, and the retransmitted flits will still contain faults. In this case, two extra clock cycles are inserted before any flit transmission. The upstream router_i first uses one clock cycle to send a control signal to downstream router_i+1 informing router_i+1 to stall an additional cycle before receiving any incoming flits. Flit will be sent via channel_i after the delay. This will relax the timing constraint on the flit transmission and reduce the probability of timing error [6] near to zero, and retransmissions are eliminated.

IV. Reinforcement Learning based Control Policy

A. Reinforcement Learning Background

Reinforcement Learning (RL) is an area of machine learning that is concerned with optimizing the behavior of autonomous agents. The RL agent interacts with an environment and takes actions that can change the state of the environment, with the aim of maximizing the total reward, or return [12].



(a) Operation Mode 0: ECC-Link_i-1 and ECC-Link_i are disabled.

(b) Operation Mode 1: ECC-Link_i-1 is disabled, and ECC-Link_i is enabled.

(c) Operation Mode 2: ECC-Link_i-1 and ECC-Link_i are enabled.
Flit pre-retransmission from router_i to router_i+1
1 clock cycle delay is inserted between pre-retransmitted flit and original flit

(d) Operation Mode 3: ECC-Link_i-1 and ECC-Link_i are enabled.
No *Flit pre-retransmission* from router_i to router_i+1;
1 clock cycle delay is inserted before any flit transmission

Fig. 3. Fault-tolerant operation modes of router_i.

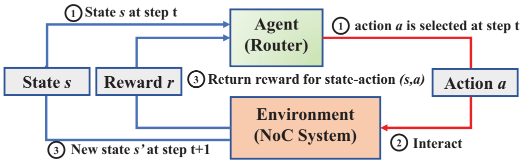

1 State s at step t
Agent (Router)
1 action a is selected at step t
State s
Reward r
3 Return reward for state-action (s,a)
Action a
Environment (NoC System)
3 New state s' at step t+1
2 Interact

Fig. 4. The interaction between the RL agent (router) and the environment (NoC) at time step t.

Agent-Environment Interaction. In the RL framework demonstrated in Fig. 4, an *agent* (NoC router) acts as a learner and a decision maker and interacts with the *environment* (entire NoC system). These interactions take place in a sequence of discrete *time steps* t = 0, 1, 2, 3, ... As shown in Fig. 4, ① in time step t, the agent (router) selects an *action* (fault-tolerant operation mode) for the current *state* and applies it at next step $t' = t + 1$. The specific action changes the environment in ②, leading to variations of the NoC attributes (*e.g.* link utilization, buffer utilization, *etc.*), which will result in a new state s' in ③. The agent observes the new state and the time step is incremented. In addition, ③ the agent also receives a *reward* r (network latency and energy efficiency). A *policy* π maps states to actions, specifying how to choose actions given the state in order to maximize the cumulative reward. For the router agent, the cumulative reward will be a function of energy, performance, and reliability over the entire sequence of actions. An RL algorithm continually evolves the policy based on the agent's past interactions with the environment.

Goal of RL Agents, Rewards and Return. In RL, the goal of an RL agent is to optimize its long-term *return*, i.e. the discounted sum of future rewards. The return at time step t is therefore defined as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad (1)$$

The variable γ (where $0 \leq \gamma \leq 1$) in this equation is a *discount rate* which determines the impact of future rewards on the total return: as γ approaches 1, the agent becomes less near-sighted by giving increasing weight to future rewards.

Action-Value Function and Q-learning. In RL, a *model* of the environment characterizes how the state of the environment changes as a result of an agent action, and the reward that the agent receives after each action. Correspondingly, RL agents compute an action-value function $Q^\pi(s, a)$ that captures the return they are expected to receive in this model of the environment if they start in state s, take action a, and follow the policy π for the remaining actions.

In this paper, we use the tabular Q-learning algorithm [12] to find the optimal Q-value function. A Q-value table is

TABLE I
FEATURES IN STATE SPACE OF EACH ROUTER.

Features (state attributes)	Description
1. Input Buffer Utilization	Number of occupied input VCs
2. Input Link Utilization	Input flits/cycle for each port
3. Output Link Utilization	Output flits/cycle for each port
4. Input NACK Rate	Percentage rate of NACK received
5. Output NACK Rate	Percentage rate of NACK sent
6. Temperature	Local router temperature ($^{\circ}C$)

initialized with zeros for all possible *(s, a)* pairs. At each time step, the Q-learning algorithm chooses actions, based on the current Q, such that, over many time steps, all actions are taken in all states. In each time step, after taking an action *a* and observing the reward *r* and new state *s′*, the action-value table entry *Q(s, a)* is changed using the following temporal difference rule:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')] \quad (2)$$

The learning rate α can be reduced over time and determines how well Q-learning will converge. Giving an appropriate value of α, Q-learning can converge to the optimal Q-value function Q* and its corresponding optimal policy π^* [12].

B. RL Problem Formulation

The proposed fault-tolerant controller is governed by RL. Per-router RL agents observe NoC system states, take actions according to the policy defined by the current Q-values, and receive system-level rewards to update the Q-values for the corresponding state-operation mode pair.

State and Action Space. A state s is a vector of system attributes (or *features*). In this paper, the state space for each router consists of several network related metrics shown in Table I. Features 1 to 5 have five separate values each, representing the ports in each of the 5 directions. The action space $A = \{a_0, a_1, a_2, a_3\}$ contains the four proposed fault-tolerant operation modes.

State-Action Mapping Table. As discussed in Section IV.A, RL-agents (routers) select actions according to the Q-values of a given state. Q-values for any state-action pair are recorded independently in one state-action mapping table per router. For example, the Q-values of state s_0 for each of the four operation modes are recorded as Q(s_0,a_0), Q(s_0,a_1), Q(s_0,a_2) and Q(s_0,a_3), as shown in Fig. 5. Some of these features are continuous values (*e.g.* link utilization, temperature), which could result in an infinite number of state-action pairs. Therefore, we discretize the parameters evenly in 5 bins or less, to keep the size of the state-action table small, so that Q-learning converges in feasible time. The discretized values for feature 1-3 and 6 are set to {0,1,2,3,4}, while the discretized values for feature 4-5 are set to {0,1,2,3}.

The bin sizes were chosen to be equal in linear space (*e.g.* link utilization) or log-space (*e.g.* NACK rate). According to observations of benchmarks' execution, operating temperature is in the $[50, 100]$ $^{\circ}C$ range, and the maximum link utilization is 0.3 flits/cycle. Those values are evenly discretized.

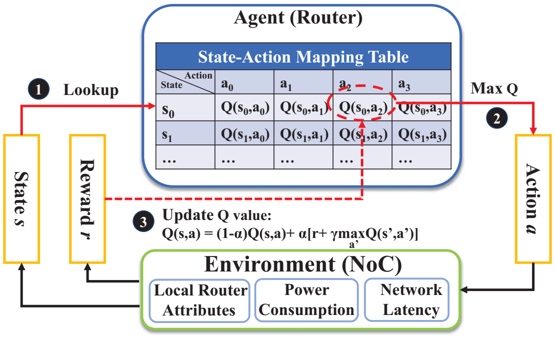


Fig. 5. Q-Learning process with procedure of updating state-action table.

Reward Function. The goal of RL agents is to maximize the total reward, which in this case implies minimizing end-to-end packet latency and the router's power consumption. Thus, we define immediate reward r in (2) for each router_i as:

$$r = [E2E\_Latency(i) \times PowerConsumption(i)]^{-1} \quad (3)$$

The $E2E\_latency(i)$ is obtained by calculating the average end-to-end latency of packets that traversed router *i*, as E2E latency is the difference between the flit injection time and the ACK injection time (same as flit arrival time). The average power consumption is monitored by NoC power modules, consisting of both static and dynamic power consumption.

C. The Working of the Proposed RL-based Control Policy

Fig. 5 shows the working of the RL-based control logic when running a benchmark. In each time step, in stage ❶, the router uses the state s (described in Section IV.B) to look up the local state-action mapping table for a matching state entry (in Fig. 5, we assume current state *s* matches state s_0 in mapping table). In stage ❷, the router selects an action *a* (one of four operation modes) which has the maximum $Q(s, a)$-value among all four possible actions in the particular state s for the next time step (we assume a_2 in Fig. 5 has the maximum Q-value). Upon taking the action *a*, the NoC system transits to a new state *s'*. In stage ❸, the NoC system provides a reward *r* (defined in Equation 5) to the router. The reward will be used to update $Q(s, a)$. Each router will repeat stages ❶ to ❸ in each time step.

Initialization of per-router controller and the state-action mapping table is as follows: RL parameters are set to default values which have been observed to achieve good results in other RL-based NoC applications [8], [13]. Q-values are initialized to 0. The learning rate α is set to 0.1, and the discount rate γ is set to 0.95. Additionally, routers have a small probability of $\epsilon = 0.1$ to select a random action instead of always taking the action with maximum Q-value in order to explore unvisited regions of the state-action space. The operation modes of all routers are initialized to mode 0.

V. EXPERIMENTAL SETUP

A. Soft Error Injection Model

Previous work [14] indicates that timing errors can be caused by NoC metrics variations. Therefore, we correlate NoC attributes and the probability of timing errors using a

TABLE II
SIMULATION PARAMETERS

# of cores	64 out-of-order CPUs @ 32nm technology
Voltage and Frequency	1.0 Volt, 2.0 GHz
NoC Parameters	8 × 8 2D Mesh, X-Y Routing 4-stage routers, 4 VCs per port
Packet Size	128bits/flit × 4 flits
Cycle delay	4 cycle to L1 cache, 8 cycle to L2 cache 160 cycle to main memory

combination of different error models and simulators, and integrate them into our network simulator to generate timing error rate for each link in an 8 × 8 2D mesh NoC topology at runtime. These models and simulators include: VARIUS [15] fault model, NoC fault model [16] and HotSpot [17] thermal model. At runtime, each router observes NoC attributes (*e.g.* V, f, link utilization) to be injected into HotSpot to obtain its temperature, and VARIUS timing error model use the temperature value to generate the probability of timing error.

B. Simulation Setup

We evaluate our proposed architecture using a modified version of the cycle-accurate network simulator *Booksim2* [18], in which we incorporate the proposed router architecture, timing error injection models and RL capabilities. We use ORION [16], a power simulator integrated in Booksim2, to evaluate the power consumption and energy efficiency. The synopsys design compiler with 32nm library is used to evaluate the area overheads. Table II shows our simulation parameters. Real-world application traces from PARSEC benchmarks, which contain packet information, injection/ejection events, and clock time stamps, are executed to analyze our framework.

We compare the performance of the proposed RL framework to three baselines: static CRC, static ARQ+ECC, and decision trees (DT). For DT, the operation modes are selected according to DT predicted error rate [6]. In DT and RL, before the testing phase of PARSEC benchmark, a pre-training phase of 1 million clock cycles and a warm-up period of 300K cycles are performed using synthetic traffic. After that, the training result of DT is no longer updated during testing phase. In RL, the temporal difference rule (2) is applied every 1K cycles. The testing phase for each benchmark lasts a full application execution time.

VI. EVALUATION AND ANALYSIS

A. Performance Analysis

Retransmission: Fig. 6 shows the number of retransmitted packets for PARSEC benchmarks as compared to other designs. It can be seen that the proposed framework achieves an average of 48% retransmission reduction over baseline. Even though ARQ+ECC achieves 33% retransmission reduction due to the ability to detect and correct single-bit errors, the proposed RL-based solution can improve upon ARQ+ECC by 15% more. Since retransmission traffic is caused by soft error, and the amount of retransmission packets reveals how the system recovers from faults. The result illustrates that our proposed fault-tolerant scheme is more powerful than other techniques for mitigating retransmission traffic, which is equivalent to providing higher fault recovery efficiency.

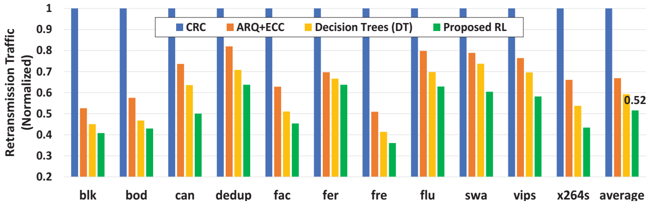


Fig. 6. Number of retransmission packets of RL as compared with other designs, normalized to CRC baseline.

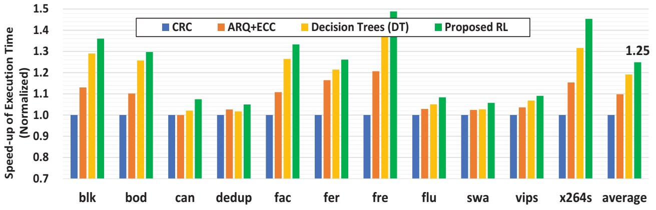


Fig. 7. Speed-up of execution time comparison, normalized to CRC baseline.

Execution Speed-up: The speed-up is obtained by calculating the full application execution time of PARSEC benchmarks for each design, as shown in Fig. 7. As can be seen in Fig. 7, the proposed architecture has an average of 1.25 × speed-up over the CRC baseline. Since the execution time depends on the amount of traffic generated while executing the benchmark, it can be deduced that the speed up can improve further for applications of higher traffic intensity.

Average End-to-End Latency: Fig. 8 shows the normalized average end-to-end packet latency of all transmitted packets. It can be seen that the proposed framework achieves an average of 55% end-to-end latency reduction over the CRC baseline. It should be noticed that ARQ+ECC achieve 30% end-to-end latency reduction over baseline. However, the proposed RL-based technique can improve upon ARQ+ECC by another 25%. This illustrates that our dynamic, proactive policy can further reduce retransmission and thus improve overall latency.

Energy Efficiency: We define energy efficiency as: $flits \times$

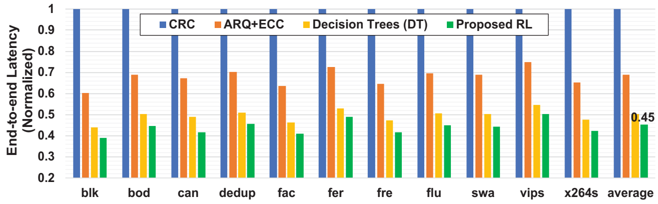


Fig. 8. Average end-to-end latency comparison, normalized to CRC baseline.

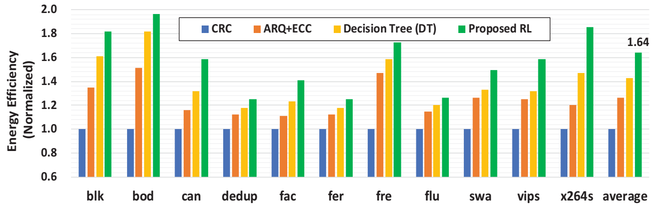


Fig. 9. Energy efficiency comparison, normalized to CRC baseline. Energy efficiency = Energy^{-1}.

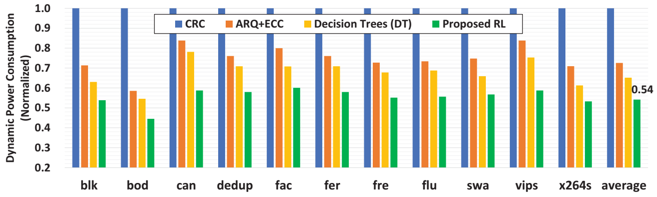


Fig. 10. Dynamic power consumption comparison, normalized to CRC baseline.

$energy^{-1}$. Energy is obtained by multiplying overall power consumption (including static and dynamic) with benchmark execution time captured through the simulator. Fig. 9 shows that the proposed framework improves energy efficiency by an average of 64% compared with CRC baseline.

Dynamic Power Consumption: The dynamic power consumption of the NoC system is related to traffic intensity. We evaluate dynamic power consumption to determine if the proposed RL design can be used for reducing traffic loads. Fig. 10 shows the dynamic power consumption analysis. It is shown that the proposed framework reduces dynamic power consumption by an average of 46% over CRC due to the RL-based design's capability of reducing retransmission traffic.

B. Overhead Analysis

Computation Overhead: The computation overhead is comprised of Q-value calculation, traversing and updating the Q-table at each time step, along with the delay time incurred during operation mode switching. For a single RL time step, the computation overhead for RL is 150ns in the worst case. However, the simulation shows such latency is overlapped due to a large time step size equals to 1K cycles.

Area Overhead: Compared with the other techniques, our RL-based design requires additional output buffers, ALU (for Q-value calculation) and SRAM storage for the Q-table for each router. We evaluate the area overhead with Synopsis Design Vision software using 32nm technology. The proposed design consumes an additional 2360 μm^2 area as compared to the conventional CRC-based router. Thus, the area overhead is 5.5%, 4.8%, and 4.5% compared with routers in the CRC, ARQ+ECC and DT approaches, respectively.

Energy Overhead: RL-based control logic can result in energy overhead compared with static fault-tolerant schemes. For each flit transmission, the proposed RL-based scheme results in a 0.16 pJ energy overhead compared with the baseline router whose energy consumption equals to 13.44 pJ. Thus, the energy overhead of RL control logic is 1.2%.

VII. Conclusions

In this paper, we propose a proactive and dynamic fault-tolerant framework using reinforcement learning (RL) to balance fault-tolerance, performance, and energy efficiency for NoC design. Specifically, the framework consists of a fault-tolerant router design with a dynamic selection of fault-tolerant schemes driven by per-router RL. Each RL agent (router) observes the NoC state, updates a control policy that dynamically enables/disables per-router error detection/correction hardware, and selects the optimal operation mode among four different fault-tolerant strategies with the dual objective of reducing overall network power consumption and latency. Evaluations show that the proposed framework achieves substantial improvements on various NoC performance metrics: 55% end-to-end latency reduction, 64% energy efficiency improvement, and 46% dynamic power reduction over the CRC baseline; and additional 10% latency reduction, 15% energy efficiency improvement, and 17% dynamic power reduction over decision trees-based technique. Area and energy overhead is 5.5% and 1.2%, respectively. Overall, this work gives a strong indication of the significant potential that reinforcement learning, and more broadly artificial intelligence, hold for the optimal design of future NoCs and multicore architectures.

Acknowledgment

This research was partially supported by NSF grants CCF-1420718, CCF1513606, CCF-1703013, CCF-1547034, CCF-1547035, CCF1540736, and CCF-1702980. We thank the anonymous reviewers for their excellent feedback.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip inteconnection networks," in *Proc. of DAC*, 2001, pp. 684–689.

[2] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Proc. of PACT*, 2011, pp. 298–309.

[3] H. Kim *et al.*, "Use it or lose it: Wear-out and lifetime in future chip multiprocessors," in *Proc. of MICRO*, 2013, pp. 136–147.

[4] C. Feng *et al.*, "Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router," *IEEE Trans. on VLSI*, vol. 21, no. 6, pp. 1053–1066, 2013.

[5] P. Poluri and A. Louri, "An improved router design for reliable on-chip networks," in *Proc. of IPDPS*, 2014, pp. 283–292.

[6] D. DiTomaso *et al.*, "Dynamic error mitigation in NoCs using intelligent prediction techniques," in *Proc. of MICRO*, 2016, pp. 1–12.

[7] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management," in *Proc. of HPCA*, 2014, pp. 308–319.

[8] Y. Bai, V. W. Lee, and E. Ipek, "Voltage regulator efficiency aware power management," in *Proc. of ASPLOS*, 2017, pp. 825–838.

[9] J. Park *et al.*, "Intelligent network-on-chip with online reinforcement learning for portable HD object recognition processor," *IEEE Trans. on Circuits and Systems I*, vol. 61, no. 2, pp. 476–484, 2014.

[10] S. Lin and D. J. Costello, *Error control coding*. Prentice Hall Englewood Cliffs, 2004, vol. 2.

[11] S. Lin and P. Yu, "A hybrid ARQ scheme with parity retransmission for error control of satellite channels," *IEEE Trans. on Communications*, vol. 30, no. 7, pp. 1701–1719, 1982.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[13] E. Ipek *et al.*, "Self-optimizing memory controllers: A reinforcement learning approach," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, 2008, pp. 39–50.

[14] K. Aisopos *et al.*, "Enabling system-level modeling of variation-induced faults in networks-on-chips," in *Proc. of DAC*, 2011, pp. 930–935.

[15] S. R. Sarangi *et al.*, "VARIUS: A model of process variation and resulting timing errors for microarchitects," *IEEE Trans. Semicond. Manuf*, vol. 21, no. 1, pp. 3–13, 2008.

[16] A. B. Kahng *et al.*, "Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proc. of DATE*, 2009, pp. 423–428.

[17] W. Huang *et al.*, "Hotspot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. on VLSI*, vol. 14, no. 5, pp. 501–513, 2006.

[18] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. of ISPASS*, 2013, pp. 86–96.