EATS: Energy-Aware Adaptive Topology Switching for NoCs

Man Wu
Keio University
Yokohama, Kanagawa, Japan
wu.man.wi5@acsl.ics.keio.ac.jp

Shaswot Shresthamali
Kyushu University
Fukuoka, Fukuoka, Japan
shaswot.shresthamali@cpc.ait.kyushu-u.ac.jp

Xiaoman Liu
Shenyang University of Technology
Shenyang, Liaoning, China
lxm@sut.edu.cn

Yuan He*
Keio University
Yokohama, Kanagawa, Japan
isaacyhe@acm.org

Abstract

The topology of a Network-on-Chip (NoC) plays a critical role in determining both its performance and power consumption. By adapting the topology to match application-specific communication demands, it is possible to improve the energy efficiency of the system. In this work, we present EATS (Energy-aware Adaptive Topology Switching), a runtime framework that dynamically reconfigures the on-chip network among multiple topologies to optimize energy usage. EATS integrates two decision engines: (i) a lightweight threshold-based heuristic controller, and (ii) a low-overhead, adaptive learning-based controller. Experimental results demonstrate that EATS can reduce NoC energy consumption by up to 47%.

CCS Concepts

• **Computer systems organization** → **Interconnection architectures**; **Reconfigurable computing**; • **Hardware** → **Interconnect power issues**.

Keywords

networks-on-chip, topology, reinforcement learning, runtime optimization, energy efficiency

ACM Reference Format:
Man Wu, Shaswot Shresthamali, Xiaoman Liu, and Yuan He. 2025. EATS: Energy-Aware Adaptive Topology Switching for NoCs. In *Great Lakes Symposium on VLSI 2025 (GLSVLSI '25), June 30–July 02, 2025, New Orleans, LA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3716368.3735175

1 Introduction

Network-on-Chips (NoCs) have emerged as a scalable and efficient alternative for overcoming the limitations of traditional microprocessor architectures [13]. By integrating multiple processing cores on a single die and connecting them via an on-chip interconnect fabric, NoCs enable packet-based communication between cores. Replacing large monolithic cores with numerous smaller ones not only improves parallelism but also enhances energy efficiency. While advances in process technology have made it feasible to incorporate an increasing number of cores onto a chip, this scaling introduces new challenges, particularly in terms of power consumption and network congestion [7] which limit the overall scalability and performance of NoCs.

Motivation: An effective strategy for improving the energy efficiency of a Network-on-Chip (NoC) is the careful selection of its topology, as it directly impacts both communication performance and power consumption. For instance, with the same number of cores, a ring topology can consume as little as one-tenth the power of a crossbar topology. However, this energy advantage may come at the cost of performance, potentially increasing communication latency by up to four times, depending on the application's traffic patterns.

Our preliminary experiments indicate that the most energy-efficient topology is highly dependent on the network's injection rate i.e., the rate at which nodes inject packets or flits into the network. Moreover, this rate can vary significantly not only between different applications, but also across different execution phases of the same application, due to diverse runtime behaviors of on-chip components. As a result, determining the optimal topology in advance is non-trivial. Relying on a static, pre-defined topology may lead to inefficient resource utilization and energy waste, especially during low-load execution phases. To achieve minimal energy usage without compromising performance, it is preferable to adapt the topology dynamically based on current network and application demands. These observations motivate the need for a runtime topology adaptation mechanism capable of adjusting to varying conditions to maintain energy-efficient NoC operation.

Proposal: To address this need, we propose a runtime **Energy-aware Adaptive Topology Switching (*EATS*)** framework that dynamically morphs the NoC topology to optimize energy efficiency. EATS leverages runtime statistics such as injection rate and energy consumption to guide topology switching decisions. The framework consists of two decision engines: a lightweight threshold-based heuristic controller and a Reinforcement Learning (RL)-based controller. The heuristic controller offers low overhead but lacks the flexibility to handle complex or dynamic workloads. In contrast, the RL-based controller *learns* topology switching policies and adapts better to diverse runtime scenarios, albeit with additional training and computation overhead. The choice of controller can be tailored to the system's performance and energy requirements. Topology switching is realized via on/off control of links and fine-grained power gating [18, 22], enabling the selective activation or deactivation of router components and interconnects. This allows the system to seamlessly transition between different topologies (e.g., ring, mesh, torus, crossbar) while maintaining a connected and functional network.

*This author is currently with RIKEN Center for Computational Science, Tokyo, Japan.

GLSVLSI '25, New Orleans, LA, USA

ACM ISBN 979-8-4007-1496-2/25/06
https://doi.org/10.1145/3716368.3735175

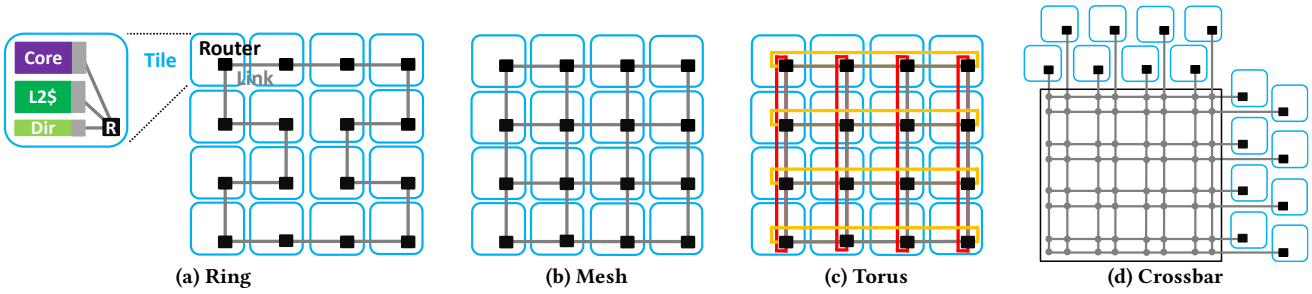


Figure 1: Four typical NoC topologies with sixteen nodes. As we move from (a) to (d), the increased connectivity generally results in higher throughput and higher energy consumption.

This work makes the following contributions:

- We propose EATS - an adaptive framework that dynamically switches the network topology of NoCs at runtime to enhance energy efficiency.
- EATS offers two controller options (threshold-based and RL-based) for runtime topology reconfiguration, each with distinct trade-offs in complexity, adaptability, and overhead.
- We provide a systematic comparison between conventional methods and both controller options, and offer practical guidelines on when each approach is most suitable.

The remainder of this paper is organized as follows. Sec. 2 provides background information relevant to this work. We present the motivation behind our approach and detail the proposed framework in Sec. 3. Sec. 4 outlines the evaluation methodology. The results and analysis are presented in Sec. 5. We finally conclude the paper in Sec. 6.

2 Background

2.1 NoC Topologies

The topology of a NoC defines the structure of routers, links, and nodes, including the number and types of interconnections. Fig. 1 illustrates four typical topologies. The topology plays a pivotal role in shaping the performance, cost, and power consumption of the network. More specifically, it dictates the amount of hops (or routers) a packet passes through and the distance between any network ends, thereby having a considerable impact on the latency and energy consumption of network traffic. Additionally, topology also outlines the total number of alternative routes between nodes, which determines the capacity of a network to distribute traffic and to meet bandwidth demands. Our evaluations show that complexity, throughput and power consumption increase from ring (Fig. 1a) to crossbar (Fig. 1d) topologies.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning framework in which an agent learns a policy to interact with its environment, aiming to maximize cumulative rewards over time [21]. In each time step, the agent observes the current state of the environment, takes an action according to its policy, receives a scalar reward, and transitions to a new state. This sequential interaction allows the agent to refine its policy through trial and error, optimizing for the long-term cumulative reward, or *return*, defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor that progressively down-weights future rewards to favor rewards at earlier time steps.

Q-learning [21] is a widely used RL algorithm that selects actions to maximize Q-values, which represent the expected return for each state-action pair under a policy π, defined as $Q^{\pi}(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a]$. In this work, the Q-values are learned via tabular Q-learning [21] because it is has very low computational costs during training and inference. In tabular Q-learning, the agent maintains a Q-table that stores Q-values for all possible state-action pairs. In the intial stage of training, the agent explores various policies and updated the Q-values in the Q-table iteratively based on the following rule:

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_a Q\left(s', a\right) - Q(s, a) \right] \tag{2}$$

where α is the learning rate, r is the reward received after taking action a from state s, and $\max_a Q(s', a)$ denotes the Q-value of the optimal action from the next state s'. After sufficient training, the Q-values will converge to their true values [21]. During inference, the optimal action for any given state is the action with the highest Q-value corresponding to that state.

2.3 Related Works

Dynamic topology reconfiguration in NoCs dates back to Wang *et al.* [22]. [6] throttle active subnets based on runtime load, while [10] select optimization strategies at runtime. Unlike these, we focus on enabling and disabling router components and links to reconfigure topology. RL-based NoC research spans both design and runtime optimization. Yin *et al.* [25] developed an RL-assisted arbiter for contention-heavy traffic; Reza *et al.* [19] used RL for dynamic routing to reduce congestion. Wang *et al.* [23] proposed a fault-tolerant RL controller to balance latency and energy. Adapt-NoC [26] applies RL to handle application-aware traffic, and Lin *et al.* [17] used deep RL for exploring routerless topologies. IntelliNoC [24] integrates Q-learning for low-latency, energy-efficient manycore NoCs. In contrast, our work targets topology switching at runtime using RL, leveraging both energy and traffic statistics.

Other energy-saving techniques span various design levels—from router-level power gating [18], DVFS [20], to system-level bandwidth-aware power provisioning [6, 11, 12], smart wiring [2], low-power buffers [8, 14, 15], and data compression [5, 9]. In contrast, we focus on topology reconfiguration at runtime by gating and de-gating network components.

3 EATS Framework

3.1 Energy Performance of Different Topologies

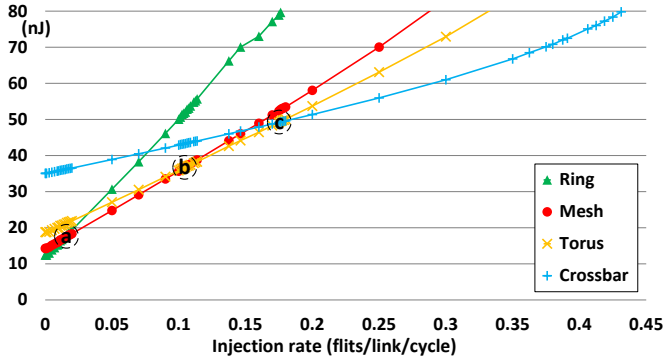


Figure 2: Energy consumption per flit versus injection rate for different network topologies under uniform random traffic conditions.

Fig. 2 shows the energy consumption per flit as a function of injection rate for four different topologies under uniform random traffic (with configurations specified in Table 1). Two key observations emerge from the figure. First, for a fixed injection rate, the choice of topology has a significant impact on energy efficiency. For example, at low injection rates (to the left of "crosspoint a"), the ring topology is the most energy-efficient. In contrast, at high injection rates (to the right of "crosspoint c"), the crossbar topology becomes the most energy-efficient.

Second, we observe a trend where topologies with higher connectivity become more energy-efficient as the injection rate increases. Notably, beyond "crosspoint c", the order of energy efficiency reverses—from ring, mesh, torus, crossbar at low loads to crossbar, torus, mesh, ring at high loads. This shift in efficiency ranking strongly motivates the need for a dynamic topology-switching framework that can adaptively select the most energy-efficient topology based on the current network injection load.

3.2 Hardware Support for Topology Switching

In this work, we assume that changing the network topology at runtime is realized through on/off links and fine-grained PG in the network [18, 22]. Such topology transitions can happen when the corresponding links and input/output ports are not occupied by any flits. We assume the routers in the NoC have three route tables, each of which serves a topology out of ring, mesh and torus while the crossbar topology is realized through an additional crosspoint crossbar switch connecting all network ends instead of the network shared by the other three topologies. When a new topology is picked, its corresponding route table will simply take over

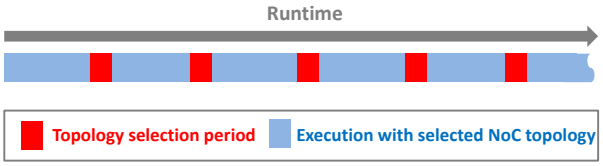


Figure 3: The epoch-based topology-switching process

while only those flits coming through links and ports that do not belong to the current topology are re-using the last route table. This way remaining traffic from previous topology settings can still be serviced without traffic loss. After topology changes, any links and ports that do not belong to this new topology can be switched off when there are no more incoming flits for them. This link and port throttling can be realized on a per-cycle basis [18]. Furthermore, flit orders are maintained at each network end by the network interface. Together with the prevention of traffic loss, this helps maintain the correct operation of the network. With our proposal, only topologies are altered at runtime, other system and network settings remain unchanged.

3.3 Epoch-based Topology Switching

In practice, the on-chip network experiences varying traffic loads (and therefore energy usage) as an application progresses through different execution phases. To address this, we design EATS as an epoch-based optimization framework that selects the most energy-efficient topology according to the current network load. EATS employs either a threshold-based heuristic or a reinforcement learning (RL)-based strategy to choose the optimal topology at the beginning of each epoch, aiming to minimize per-flit energy consumption. As the application executes, traffic statistics from the current epoch are analyzed to guide the topology selection for the next epoch. In essence, the application's execution is partitioned into multiple stages (epochs), and at each stage, EATS dynamically selects the most suitable topology for the upcoming phase using one of the two optimization strategies. This iterative approach enables EATS to adapt continuously to workload variations and maintain high energy efficiency throughout the runtime.

As illustrated in Fig. 3, execution begins with the selection of an initial topology (mesh in our case) and the application runs for one epoch while network statistics are collected. When the program reaches the topology selection phase (highlighted in red in Fig. 3), the EATS framework broadcasts an update to all routers to begin using a new routing table. This transition period ends once all in-flight flits reach their respective destinations. The duration of this period may vary depending on the volume of remaining network traffic. Afterward, the new topology is applied by activating or deactivating specific links, and the next epoch begins. This iterative process continues throughout the application's execution. Two key metrics, injection rate and energy consumption, are recorded during each epoch. They are used by EATS to guide topology selection. We experiment with epoch sizes of 10,000, 100,000, and 1,000,000 cycles, noting that larger epochs reduce switching overhead, while smaller epochs enable finer-grained adaptation at the cost of higher overhead.

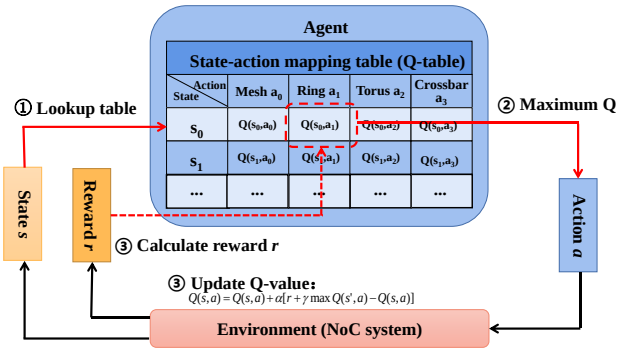


Figure 4: Updating the Q-table using RL

3.4 EATS Decision Engines

3.4.1 The RL Decision Engines. To dynamically adjust the network topology with our RL-driven approach, we define its state, action, agent, and reward as follows:

State Definition: At each topology switching point, routers collect internal network statistics to determine the state. The system state is defined using one of the following two key metrics: (i) average energy consumption per flit, or (ii) network injection rate, both of which are derived from NoC router logs. As these metrics are continuous, they are discretized into predefined ranges to enable compatibility with the tabular Q-learning model.

Action Definition: The agent's action is to select one of four network topologies: mesh, ring, torus, or crossbar, represented by the action set$\{a_0, a_1, a_2, a_3\}$ correspondingly.

Reward Definition: The goal of our RL agent is to maximize the long-term reward which guides the action selection and eventually determines the energy consumption of network flits. We use *energy_consumption_per_flit* as the reward function, defined as:

$$\begin{aligned} Reward &= -energy\_consumption\_per\_flit \\ &= -Power \times Network\_latency\_per\_flit \end{aligned} \tag{3}$$

In each cycle, the agent observes the current state and selects the action with the highest Q-value from the Q-table (i.e., the topology with the highest Q-value). The network is then reconfigured to the chosen topology. Afterwards, a reward is computed which the agent uses to refine its policy through the Q-learning update.

Training: Fig. 4 illustrates how we apply the RL to switch the topology. In step ①, the agent looks up the state-action mapping table corresponding to the observed state s_0. In step ②, the agent selects a corresponding topology a from a_0, a_1, a_2, a_3, which has the maximum $Q(s_0, a)$ value. In step ③, after selecting the action a, the network topology is configures. In the next epoch, the the reward r is calculated using Eq. (3), which depends on the performance of the selected topology (action). Subsequently, the Q-value $Q(s_0, a)$ (i.e., the long-term return associated with the chosen state-action pair) is updated based on the received reward using Eq. (2). Finally, the updated Q-table is used in the next iteration, allowing the agent to continually improve its decision-making process over time. During learning, we use an ϵ-greedy policy. Under this policy, the agent occasionally selects a random action with a probability of ϵ, ensuring that the learning process explores different actions, which prevents it from converging prematurely to a suboptimal policy.

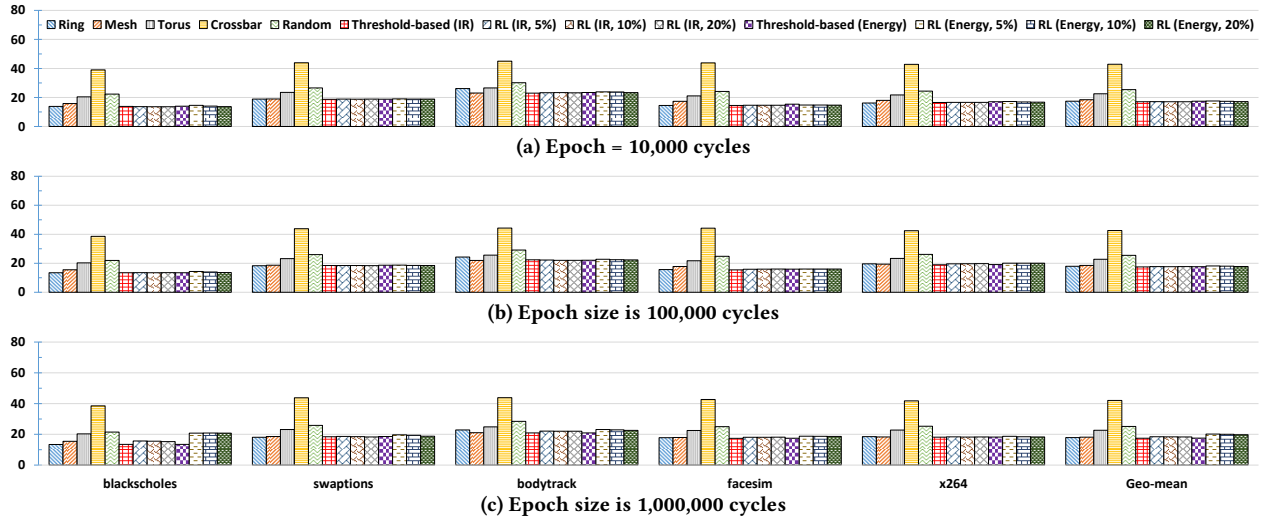
The agent starts with an uninitialized (zeroed) Q-table. We dedicate early epochs (such as 5%, 10%, and 20% of the execution time) for exploratory learning, during which the agent updates its policy via trial-and-error. After this phase, the Q-table is frozen, and the agent's greedy policy is evaluated over the remaining epochs without further updates.

3.4.2 The Threshold-Based Engine. In addition to RL-driven control, we propose an alternative method threshold-based heuristic engine for EATS. The threshold values for topology change are determined using the crosspoints in Fig. 2. The figure shows the energy consumption per flit versus injection rate across four distinct topologies under uniform random traffic. At an injection rate of 0.025 (crosspoint a), energy consumption is the same for the ring and torus topologies. Similarly, at an injection rate of 0.110 (crosspoint b), the mesh and torus topologies exhibit equivalent energy consumption, and at 0.175 (crosspoint c), the crossbar and torus topologies demonstrate matching energy profiles. Therefore, using these crosspoints, we can readily identify an appropriate topology-switching action. For example, we select the ring topology when the injection rate is below 0.025. For injection rates between 0.025 and 0.110, mesh is better. When the injection rate ranges from 0.110 to 0.175, torus is preferred. For injection rates exceeding 0.175, crossbar excels. This threshold-based method requires a very simple controller, which is implemented and connected to all routers. This controller has preset values of injection rates and per-flit energy values as the thresholds. With these thresholds, we can easily determine a topology-switching action. However, these thresholds are derived from uniform random traffic, which may not be optimal for complex situations.

With both approaches (the RL-driven and the threshold-based), it is possible to balance cost and system performance while minimizing per-flit energy consumption in the network, providing flexibility to meet specific requirements. With our framework, we assume that the topology selection periods allow sufficient time for the RL agent and the threshold-based controller to communicate with routers for gathering statistics and switching the topology.

Table 1: Evaluation Hardware Platform Configuration

Number of nodes	16
Topologies	ring, mesh, torus, and crossbar
Processor	4 GHz, timing simple
L1 I/D cache	32 KB per processor, 4-way set associative, 4 cycles per access
L2 cache	256 KB per bank, 16-way set associative, 40 cycles per access
Cache line	64 Bytes
Main memory	8 GB, 120 cycles per access
Coherence protocol	MESI 2-level
Link	128-bit, 1 cycle traversal
Packet	128-bit control, 640-bit data
Router	4 GHz, 4-stage pipeline
Virtual channel	4 per virtual network
Virtual network	3 per physical link
Process technology	22 nm
Vdd	1 V



(a) Epoch = 10,000 cycles

(b) Epoch size is 100,000 cycles

(c) Epoch size is 1,000,000 cycles

Figure 5: Energy consumption per flit with fixed topologies and proposed schemes with the original (1x) injection rate

Table 2: Evaluation Parameters

Applications	*blackscholes*, *bodytrack*, *facesim*, *swaptions*, and *x264*
Input sizes of the applications	small for *blackscholes*; medium for *bodytrack*, *swaptions*, and *x264*; large for *facesim*
Injection rates	1x, 2x, and 4x
Epoch sizes (cycles)	10,000; 100,000; 1,000,000
Percentage of epochs for exploration	5%, 10%, and 20%
RL model parameters	$\alpha = 0.1$, $\gamma = 0.9$, and $\epsilon = 0.01$

4 Evaluation Methodology

In this paper, trace-based simulations are used to evaluate the performance and energy consumption of NoCs. First, we collect on-chip traffic with the help of gem5 [4] computer architecture simulators. This traffic is then replayed in GARNET [1], a cycle-accurate network model, to retrieve the network statistics (with various performance-related metrics). Afterwards, such network statistics are fed to McPAT [16] power modeling framework to evaluate the power consumption of the network. In addition, both of our optimization approaches are simulated with the tool implemented by Reza *et al.* [19]. We have modified their tool to implement our threshold-based approach and repurposed their RL implementation for our experiments. Configurations of the evaluated hardware platform are summarized in Table 1 while the five PARSEC workloads [3] and evaluated parameters are presented in Table 2. To speed up the evaluation, we picked the "Timing Simple" CPU model in gem5. It is relatively simple but we have condensed the traces for two and four times to increase the injection to the network (to mimic more powerful CPU models).

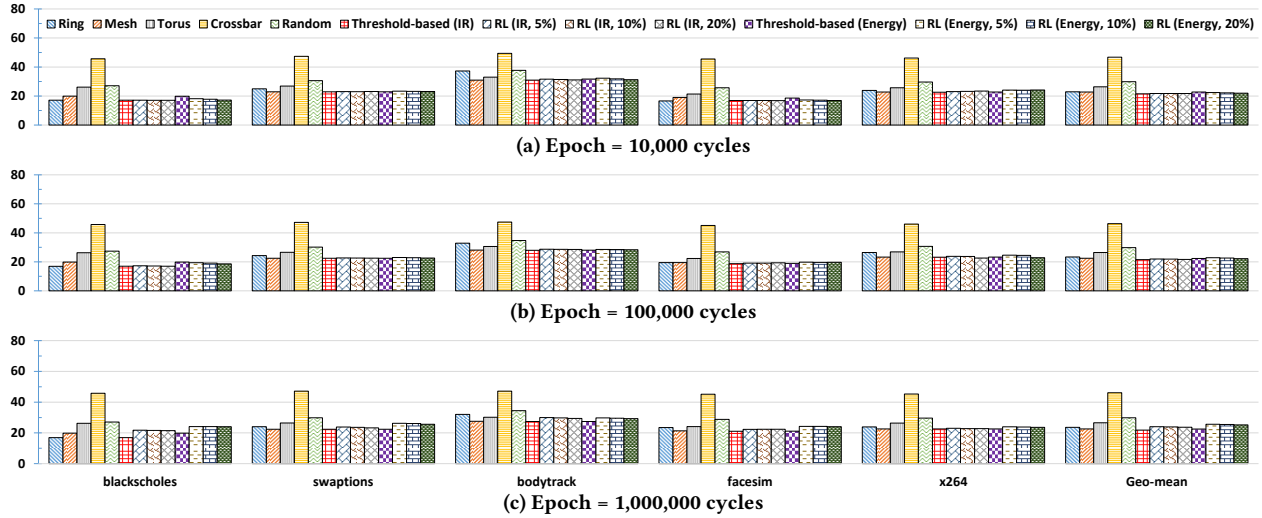
The thresholds in our proposals are derived from the uniform random synthetic traffic as mentioned earlier. The RL model used has a learning rate of $\alpha = 0.1$, a discount rate of $\gamma = 0.9$, and an exploration rate of $\epsilon = 0.01$. These parameters are carefully picked based on related studies [17, 19, 23–26]. Our RL-driven approach makes use of statistics extracted from GARNET and McPAT outputs, including power consumption, simulation time, average network latency, and the number of flits.

5 Results

The evaluation results are shown in Fig. 5, Fig. 6, Fig. 7, and Fig. 8. We conduct simulations across four fixed topologies (ring, mesh, torus, and crossbar), a randomly switching baseline, and eight configurations of our proposed EATS framework. Among these, two configurations use the threshold-based approach, which adapts the topology based on either injection rate or per-flit energy consumption. These are labeled as "Threshold-based (IR)" and "Threshold-based (Energy)". The remaining six configurations employ the RL-based approach, also adapting based on either injection rate or energy consumption, but with varying levels of exploration during learning. These are denoted as "RL (IR, 5%)", "RL (IR, 10%)", "RL (IR 20%)", "RL (Energy, 5%)", "RL (Energy, 10%)", and "RL (Energy, 20%)".

Within Fig. 5, Fig. 6, and Fig. 7, there are three sub-figures, each of which corresponds to an epoch size. Fig. 5 represents the originally collected network trace while with Fig. 6 and Fig. 7, we have condensed the trace so that traffic to the network is injected within half or one-quarter of the original time.

It can be observed that both our proposals (threshold-based and RL-based) perform well regardless of the amount of injection but the RL-based approach is marginally better than the threshold-based approach with higher injections. On average, under 4x injection and 10% being the amount of exploratory learning, both proposed approaches reduce energy consumption by 4.7% to 23.6% compared to the best topology (Mesh), by 23.3% to 23.6% compared to the random case, and by up to 47.4% compared to the worst topology (Crossbar). In the `Blackscholes` benchmark at the 1x injection rate, the proposed approaches achieve energy reductions of 13.3% to 9.7% compared to Mesh, of 38.7% to 36.2% compared to the random case, and up to 65.1% compared to Crossbar. Overall, both demonstrate competitive outcomes compared to fixed topologies. Notably, the threshold-based one using injection rate as the metric proves to

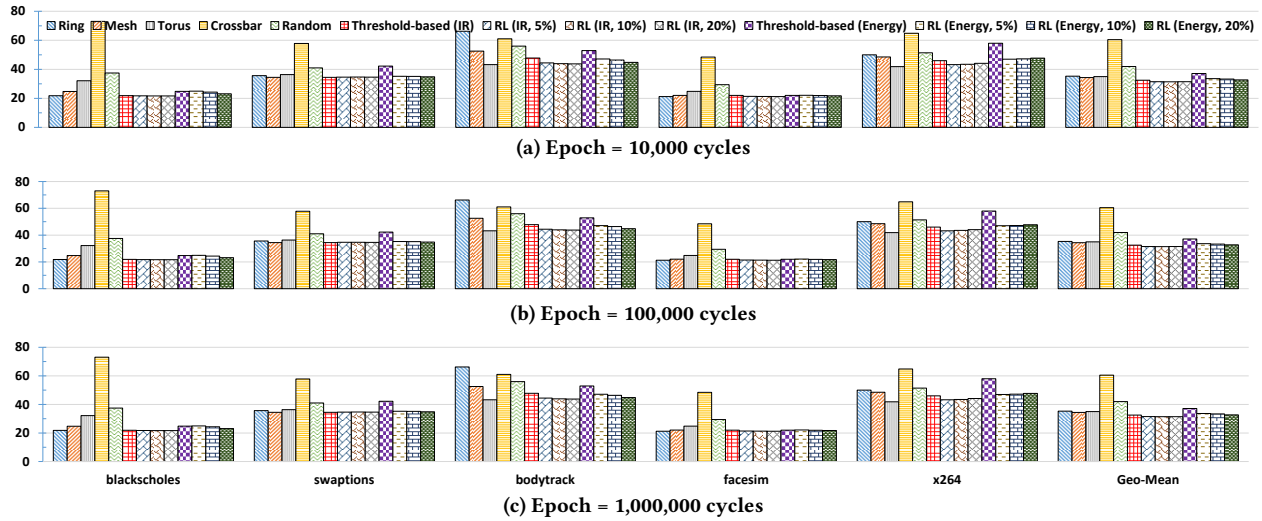


(a) Epoch = 10,000 cycles

(b) Epoch = 100,000 cycles

(c) Epoch = 1,000,000 cycles

Figure 6: Energy consumption per flit with fixed topologies and proposed schemes with 2x injection rate



(a) Epoch = 10,000 cycles

(b) Epoch = 100,000 cycles

(c) Epoch = 1,000,000 cycles

Figure 7: Energy consumption per flit with fixed topologies and proposed schemes with 4x injection rate

be the most efficient. Furthermore, using injection rate as the input outperforms energy consumption as the input to optimize the network energy efficiency.

Regarding epoch size, we can observe that our two approaches perform slightly better with smaller epochs, especially when injection load is low (as in Fig. 5 and Fig. 6). This is reasonable since smaller epochs result in more adaptive changes. But in reality, too small epoch sizes may incur large adaptation overheads. For the amount of exploratory learning, we notice that the RL-based approach behaves similar with 5%, 10% and 20% of the execution time dedicated to training. This indicates that setting 5% as the amount of exploratory learning may already suffice.

Moreover, there are two phenomena we learn from these results. First, a fixed topology cannot guarantee the best energy efficiency. As ring, mesh, or torus topology can perform better than other fixed topologies for different workloads under various injection factors. This emphasizes the motivation of our work, that is, a topology may be suitable for an epoch when a workload runs, but not all the epochs. Second, our proposals work to some degree but the results differ among workloads, approaches, inputs, and injection factors. In general, they perform very similarly when the injection factor is low (as in Fig. 5). We can see that both approaches can help produce on-par energy consumption per flit with the best fixed topology.

Last but not least, with Fig. 8, we have presented the rewards over time in the RL-driven case with injection rate as the input, four times the injection, and an epoch size of 100 thousand cycles. It can be seen that the rewards stabilized at different time for these five benchmarks, which means different workloads require different amount of efforts (10%~20% of the execution time) for the RL agent to be trained and there are cases where additional training may be necessary during different phases of the workloads.

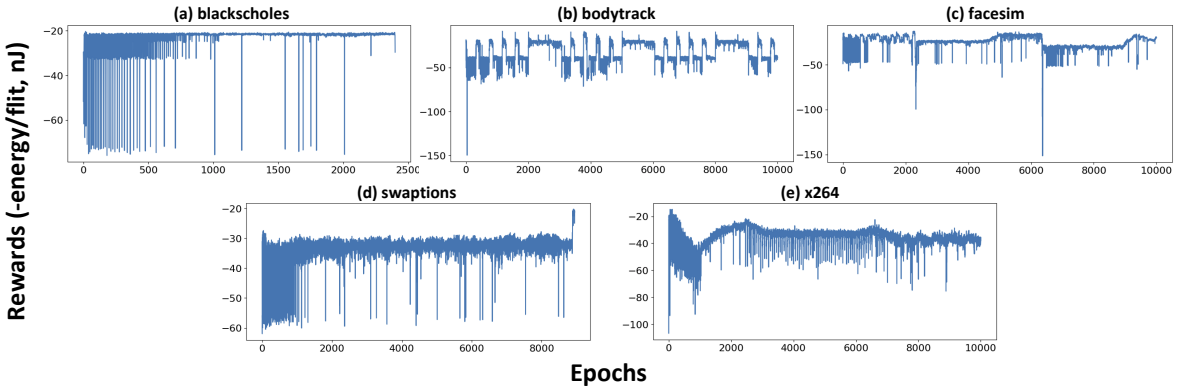


Figure 8: Rewards over different epochs at runtime under 4x injection rate with an epoch size of 100,000 cycles

6 Conclusion

Beyond the primary evaluation results, we highlight several key insights. First, threshold-based methods perform well in typical scenarios and are nearly on par with RL-based approaches under moderate loads. However, their effectiveness declines under high injection rates, suggesting that hybrid or adaptive schemes may be necessary in dynamic environments. Second, RL-based methods, while promising, face three key challenges: (i) computational and hardware overheads, and (ii) the need for training period. To address these, we advocate for offline pre-training and simplified models, which reduce cost but may limit adaptability during execution. Third, hardware support for topology switching incurs overhead. Sleep transistors (for PG), multiple routing tables, and possibly an additional crosspoint switch (for crossbar support) are needed. In practice, a subset of topologies, such as ring, mesh, and torus, offers a more feasible trade-off between flexibility and implementation cost. Fourth, the input metrics we have used i.e., the injection rate per link and energy consumption, are global and topology-agnostic, making them suitable for a system with a single active topology. Finer-grained, spatially adaptive schemes are possible but fall outside the scope of this work. Finally, while EATS currently optimizes network-level energy, it can be extended to incorporate system-wide metrics, enabling more holistic optimization across the chip.

In conclusion, this work demonstrates that dynamic topology adaptation, either through a lightweight threshold-based method or a more adaptive RL-based approach, can significantly reduce NoC energy consumption compared to fixed static topologies. While RL-based strategies show strong potential, especially under diverse workloads, their practicality hinges on reducing computational overhead. Overall, runtime topology switching presents a promising direction for energy-aware NoC design.

Acknowledgments

This work was supported, in part, by JSPS KAKENHI with Grants JP24K20843, JP22K21285, and JP20K23315, and by the Okawa Foundation for Information and Telecommunications under Grant 21-04, all from Japan.

References

[1] N. Agarwal et al. 2009. GARNET: a detailed on-chip network model inside a full-system simulator. In *Proc. of the ISPASS'09*. 33–42.
[2] F. Alazemi et al. 2018. Routerless Network-on-Chip. In *Proc. of the 24th HPCA*. 492–503.
[3] C. Bienia et al. 2008. PARSEC vs. SPLASH-2: a quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *Proc. of IISWC'08*. 47–56.
[4] N. Binkert et al. 2011. The Gem5 Simulator. *ACM SIGARCH CAN* 39, 2 (Aug 2011), 1–7.
[5] R. Das et al. 2008. Performance and power optimization through data compression in Network-on-Chip architectures. In *Proc. of 14th HPCA*. 215–225.
[6] R. Das et al. 2013. Catnap: Energy Proportional Multiple Network-on-Chip. In *Proc. of 40th ISCA*. 320–331.
[7] H. Esmaeilzadeh et al. 2011. Dark Silicon and the End of Multicore Scaling. In *Proc. of 38th ISCA*. 365–376.
[8] Y. Gao et al. 2022. Traffic-Aware Energy-Efficient Hybrid Input Buffer Design for On-Chip Routers. In *Proc. of 15th MCSoC*. 395–401.
[9] Y. He et al. 2012. Adaptive data compression on 3D network-on-chips. *IPSJ Online Transactions* 5, 1 (Jan 2012), 13–20.
[10] Y. He et al. 2015. Runtime Multi-Optimizations for Energy Efficient On-Chip Interconnections. In *Proc. of 33rd ICCD*. 455–458.
[11] Y. He et al. 2016. Opportunistic Circuit-Switching for Energy Efficient On-Chip Networks. In *Proc. of 24th VLSI-SoC*. 1–6.
[12] Y. He et al. 2020. Energy-Efficient On-Chip Networks through Profiled Hybrid Switching. In *Proc. of 30th GLSVLSI*. 241–246.
[13] Y. Hoskote et al. 2007. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro* 27, 5 (Nov 2007), 51–61.
[14] H. Jang et al. 2012. A Hybrid Buffer Design with STT-MRAM for On-Chip Interconnects. In *Proc. of 6th NOCS*. 193–200.
[15] C. Li et al. 2015. A compact low-power eDRAM-based NoC buffer. In *Proc. of ISLPED'15*. 116–121.
[16] S. Li et al. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. of 42nd MICRO*. 469–480.
[17] T.R. Lin et al. 2020. A deep reinforcement learning framework for architectural exploration: A routerless NoC case study. In *Proc. of 26th HPCA*. 99–110.
[18] H. Matsutani et al. 2010. Ultra Fine-Grained Run-Time Power Gating of On-Chip Routers for CMPs. In *Proc. of 4th NOCS*. 61–68.
[19] M.F. Reza et al. 2021. Reinforcement learning enabled routing for high-performance networks-on-chip. In *Proc. of ISCAS'21*. 1–5.
[20] L. Shang et al. 2003. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of 9th HPCA*. 91–102.
[21] R.S. Sutton et al. 2018. *Reinforcement learning (2ed): an introduction*. MIT Press, Cambridge, MA, USA.
[22] D. Wang et al. 2008. A link removal methodology for Networks-on-Chip on reconfigurable systems. In *Proc. of FPL'08*. 269–274.
[23] K. Wang et al. 2019. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *Proc. of DATE'19*. 1166–1171.
[24] K. Wang et al. 2019. IntelliNoC: A holistic design framework for energy-efficient and reliable on-chip communication for manycores. In *Proc. of 46th ISCA*. 589–600.
[25] J. Yin et al. 2020. Experiences with ML-Driven Design: A NoC Case Study. In *Proc. of 26th HPCA*. 637–648.
[26] H. Zheng et al. 2021. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *Proc. of 27th HPCA*. 723–735.