# NOC Optimization Using LESSON Algorithm: Design Document

## 1. Introduction

This document presents a detailed design for optimizing the Network-on-Chip (NOC) architecture using the LESSON (Learning Enabled Sleepy Storage Links and Routers in NoCs) algorithm. The optimization aims to achieve the following criteria:

- Minimize latency (≤ min_latency)
- Maximize bandwidth (at 95% of max_bandwidth)
- Maintain buffer occupancy around 90%
- Limit throttling to only 5% of the time

The LESSON algorithm, which combines principles of reinforcement learning (RL) with innovative techniques tailored for NOC optimization, is well-suited for this problem due to its ability to handle continuous state and action spaces, as well as delayed rewards effectively.

## 2. NOC Architecture with Sleepy Link Storage (SLS) Units

The proposed NOC architecture incorporates Sleepy Link Storage (SLS) units, which enable both link reversal and power-gating capabilities. These SLS units consist of the following components:

- An inverter
- Additional transistors for storage
- Transmission gates for bidirectional data transmission

Power-gating is achieved by activating a high-threshold voltage transistor connected to the SLS units.

### 2.1. SLS Channel and Link Configuration

SLS channels are interconnected with routers, forming links. Each link consists of multiple SLS units connected in series. The decision to reverse links or power-gate them is made based on link utilization predictions provided by the LESSON algorithm.

Two controllers are responsible for managing link configurations:

1. **Direction Controllers**: Allocate directions to links based on predicted utilization levels.

2. **Power-gating Controllers**: Determine which links to power-gate based on predicted utilization levels.

### 2.2. Modular Crossbar Design

To reduce power consumption, the crossbar within each router is split into smaller modular crossbars. Each modular crossbar represents a routing quadrant and is independently power-gated based on link utilization predictions from the LESSON algorithm.

### 2.3. Router Power-Gating

In addition to the modular crossbars, other components within the routers, such as MUXs, DEMUXs, escape buffers, and crossbars, are also power-gated based on predicted link utilization levels from the LESSON algorithm. This further reduces static power overhead within the NOC.

## 3. LESSON Algorithm Integration

The LESSON algorithm is selected for this optimization problem due to its suitability for handling continuous state and action spaces, as well as its ability to effectively manage delayed rewards, which are critical aspects of NOC optimization.

### 3.1. Training Procedure

The training procedure for the LESSON algorithm involves the following steps:

1. **Data Collection**: Simulate the NOC with various parameter configurations and collect data on link utilization, buffer occupancy, throttling, and other relevant metrics.
2. **Preprocessing**: Normalize the collected states and rewards to facilitate training.
3. **Model Architecture**: Design neural networks for the policy (actor) and value (critic) networks, tailored to the LESSON algorithm and the NOC optimization problem.
4. **Training**: Train the LESSON agent using the collected data, optimizing for the defined reward function based on the specified optimization criteria.
5. **Hyperparameter Tuning**: Fine-tune hyperparameters such as learning rate, discount factor, and network architecture to achieve optimal performance.
6. **Evaluation**: Evaluate the trained agent on test scenarios to ensure it meets the defined optimization criteria.

### 3.2. Evaluation and Deployment

1. **Evaluation**: Test the trained LESSON agent on various NOC configurations to assess its performance and validate that it meets the specified optimization criteria.

2. **Deployment**: Deploy the trained LESSON agent in the NOC system for real-time optimization. Continuously monitor NOC performance and fine-tune the RL agent based on real-world data.

# 4. Advantages of LESSON over Alternative Algorithms

The LESSON algorithm is preferred over alternative algorithms like Random Forest and Support Vector Regression (SVR) for the following reasons:

1. **Dynamic Adaptability**: The LESSON algorithm adapts to changing network conditions and adjusts link directions and power-gating strategies accordingly, ensuring optimal performance under varying scenarios.
2. **Continuous Action Spaces**: LESSON handles continuous action spaces effectively, allowing for fine-grained control over NOC parameters such as buffer sizes, arbitration weights, and throttling.
3. **Delayed Rewards Handling**: The RL framework of LESSON efficiently handles delayed rewards, which is crucial for optimizing NOC performance over time.

Random Forest and SVR, being supervised learning algorithms primarily used for classification and regression tasks, may not handle the dynamic and continuous nature of NOC optimization problems effectively. Additionally, they may struggle with handling delayed rewards and continuous action spaces, which are essential for NOC optimization.

# 5. Time Complexity Analysis

- **LESSON Algorithm**: The time complexity of the LESSON algorithm depends on the size of the NOC topology and the number of parameters involved. The training process, which involves collecting data from simulations, preprocessing, training neural networks, and hyperparameter tuning, is computationally intensive. However, once trained, the inference phase for real-time optimization has lower time complexity as it involves making decisions based on learned policies.
- **Random Forest and SVR**: Both Random Forest and SVR have higher time complexity compared to RL algorithms like LESSON, especially for large datasets and high-dimensional feature spaces. Training these models involves constructing decision trees (for Random Forest) or solving complex optimization problems (for SVR), which can be computationally expensive.

# 6. Conclusion

This design document outlines the proposed approach for optimizing the NOC architecture using the LESSON algorithm. By integrating the LESSON algorithm into the RL framework, the system can adapt to changing conditions and dynamically adjust buffer sizes, arbitration weights, throttling, and power-gating strategies to optimize NOC performance in real-time. The

LESSON algorithm provides a comprehensive solution for NOC optimization, leveraging reinforcement learning principles with innovative techniques tailored for NOC architectures.

# 7. Proof of Correctness

Algorithm 1 Offline learning with the ID3 algorithm.

```
ID3(Training data D, Attribute A):
    if all samples in D have the same label:
        return a leaf node with that label

    if levels in the tree equals 3:
        return a leaf node with label chosen from a majority vote

    let X ∈ A be the attribute with the largest information gain
    let R be a tree root labeled with attribute X

    let D1, D2, ..., Dk be the partition produced by splitting D on
attribute X

    for each Di ∈ D1, D2, ..., Dk:
        let Ri = ID3(Di, A - {X})
        add Ri as a new branch of R

    return R
```

**Assumptions:**

1. The decision tree algorithm used for predictions is the ID3 algorithm, which is a well-established and widely used algorithm for building decision trees.
2. The training data used for building the decision trees is representative of the actual NOC behavior and covers a diverse range of scenarios.
3. The information gain metric used by the ID3 algorithm is an appropriate measure for selecting the best attribute at each node of the decision tree.
4. The termination condition of limiting the decision tree depth to three levels is sufficient for making accurate predictions while maintaining low overhead during testing.

**Lemma 1:** The ID3 algorithm, when provided with sufficient representative training data and an appropriate attribute selection metric (e.g., information gain), will construct a decision tree that accurately captures the underlying patterns and relationships between the input attributes and the target variable (link utilization or traffic load).

**Proof:** The ID3 algorithm is a greedy algorithm that recursively partitions the training data based on the attribute that provides the highest information gain (or other appropriate attribute selection metric) at each level of the tree. By selecting the most informative attribute at each node, the algorithm effectively reduces the entropy (or uncertainty) of the remaining subsets of data. This greedy approach has been shown to converge to an optimal decision tree when provided with sufficient representative training data (Quinlan, 1986).

**Lemma 2:** The decision trees constructed by the ID3 algorithm, when used for testing new samples, will correctly classify the samples if the samples are drawn from the same distribution as the training data and the decision tree has accurately captured the underlying patterns.

**Proof:** Once the decision tree is constructed, testing a new sample involves traversing the tree from the root node to a leaf node by evaluating the attribute values of the sample at each internal node. If the decision tree accurately captures the underlying patterns in the data, and the new sample is drawn from the same distribution as the training data, the traversal will lead to the correct leaf node, which represents the correct classification or prediction for the sample.

**Theorem:** The proposed approach of using decision trees constructed by the ID3 algorithm for predicting link utilization and traffic load, followed by adjusting link directions and power-gating strategies based on these predictions, will correctly optimize the NOC performance according to the specified criteria (minimize latency, maximize bandwidth, maintain buffer occupancy, and limit throttling) if the following conditions hold:

1. The training data used for building the decision trees is representative of the actual NOC behavior and covers a diverse range of scenarios.
2. The information gain metric used by the ID3 algorithm is an appropriate measure for selecting the best attribute at each node of the decision tree.
3. The termination condition of limiting the decision tree depth to three levels is sufficient for making accurate predictions while maintaining low overhead during testing.
4. The link direction and power-gating controllers correctly interpret and act upon the predictions made by the decision trees.

**Proof:** By Lemma 1, the ID3 algorithm constructs decision trees that accurately capture the underlying patterns and relationships between the input attributes (e.g., current NOC configuration) and the target variables (link utilization and traffic load). By Lemma 2, these decision trees will correctly predict the link utilization and traffic load for new samples (NOC configurations) drawn from the same distribution as the training data.

Given these accurate predictions, the link direction and power-gating controllers can make informed decisions about adjusting link directions and power-gating strategies to optimize NOC performance. Specifically:

1. By accurately predicting link utilization, the power-gating controllers can identify underutilized links and power-gate them, reducing static power consumption while maintaining necessary bandwidth.

2. By accurately predicting traffic load, the link direction controllers can reverse link directions to accommodate changes in traffic patterns, minimizing latency and maximizing bandwidth utilization.
3. By adjusting link directions and power-gating strategies based on these predictions, the overall NOC performance can be optimized to meet the specified criteria of minimizing latency, maximizing bandwidth, maintaining buffer occupancy, and limiting throttling.

Therefore, if the stated conditions hold, the proposed approach of using decision trees for predictions and adjusting link configurations and power-gating strategies accordingly will correctly optimize the NOC performance according to the specified criteria.