Microprocessors and Microsystems 90 (2022) 104485

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

DeepNR: An adaptive deep reinforcement learning based NoC routing algorithm

Reshma Raj R.S. [a], Rohit R. [a], Mushrif Shaikh Shahreyar [a], Akash Raut [b], Pournami P.N. [a], Saidalavi Kalady [a], Jayaraj P.B. [a,*]

[a] *Department of Computer Science and Engineering, National Institute of Technology Calicut, Kerala 673601, India*
[b] *Department of Electrical Engineering, National Institute of Technology Calicut, Kerala 673601, India*

ARTICLE INFO

Keywords:
Network-on-Chip
Routing algorithm
Deep reinforcement learning
Packet latency
Throughput

ABSTRACT

Network-on-Chip (NoC) has become a cost-effective communication interconnect for Tiled Chip Multicore Processor systems. The communication between cores is done through packet exchange. As the computational intensity of applications increases, the amount of packet exchange between cores will also increase. The improper routing of these packets will result in high congestion thereby degrading the system performance. This marks the need for congestion-aware routing in NoC. In the real world, the applications running in NoC create diverse traffic, which in turn creates challenges in routing. Such challenges have resulted in more researchers relying on machine learning algorithms to tackle them. However, the issues pertaining to storage overhead and packet latency prevail in such methodologies. This paper presents an adaptive routing algorithm DeepNR, which uses a deep reinforcement learning approach. The proposed approach considers network information for state representation, routing directions for actions, and queuing delay for reward function. Experiments carried out on synthetic as well as real-time traffics to demonstrate the effectiveness and efficiency of DeepNR using the Gem5 simulator. The results obtained for DeepNR indicate a reduction of up to 21.25% and 44% in overall packet latency under high traffic conditions on real and synthetic traffic respectively, as compared to the existing approaches. Also, DeepNR achieves a throughput of above 90% in both the traffic scenarios.

1. Introduction

The ever increasing demands associated with the latest developments in the field of science and technology have prompted the researchers to develop and deploy hardware with high processing capabilities. As a first step towards this, researchers focused on increasing the performance of single-core processors which in turn results in an uncontrolled and unbearable rise in power consumption and heat [1]. From 2002 onwards, there happened a paradigm shift from the single monolithic microprocessor to multi-core architectures and then to current many-core architectures [2]. The transformed paradigm, in which complex heterogeneous functional elements are integrated on a single chip, requires bus structures for chip communication [3]. Since the number of processing elements increased to hundreds and then to a few thousand, the traditional bus communication lacks scalability and predictability and is not capable to keep up with the increasing requirements regarding performance, and power [4]. This raised a challenge to the researchers to come up with new technologies for interconnecting processors with each other. Thus, an optimistic solution called Networks-on-Chip (NoC) emerged, which is the most effective, scalable, and flexible chip interconnection mechanism for multi-core and many-core designs [5].

In the semiconductor industry, it is anticipated that the number of processors in an Integrated Chip (IC) will increase in the forthcoming years. As per International Technology Roadmap for Semiconductors (ITRS) report-2015, the increasing demand for information processing will drive 30 times increase in the number of processing cores by 2030 [6]. Nowadays, the high level of parallelism demonstrated by multicore systems running with complex applications has resulted in high communication between cores. This would potentially cause an increase in the number of packets in the NoC network. Hence, NoC requires new requirements such as different service classes, effective application mapping, and efficient algorithms for routing and congestion management [7]. The performance of NoC mainly depends on the efficiency of routing techniques. An efficient routing technique should be capable of distributing packets from source to destination through less congested paths. In general, there are two main classes of NoC routing algorithms — deterministic and adaptive. Deterministic routing algorithms route packets along a certain fixed path whereas

* Corresponding author.
E-mail address: jayarajpb@nitc.ac.in (Jayaraj P.B.).

https://doi.org/10.1016/j.micpro.2022.104485
Received 16 August 2021; Received in revised form 8 January 2022; Accepted 10 February 2022
Available online 23 February 2022

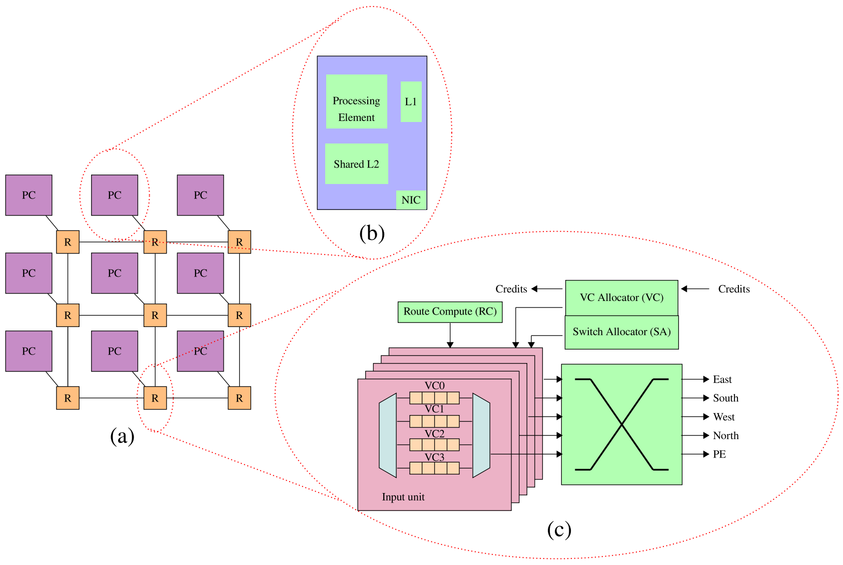

Fig. 1. NoC architecture.

in adaptive routing algorithms, the intelligent routers are the ones responsible for finding a path based on network conditions. A significant amount of works has been reported in the area of NoC routing [8,9]. Unfortunately, these existing techniques cannot withstand the amount and diversity of traffic as the number of cores and applications increased in the chip. To deal with the diverse nature of large traffic, researchers started introducing Machine Learning (ML) techniques to on-chip technology [10]. Learning traffic behavior and optimizing latency, in response to run-time changes, are promising features of NoC for applying ML. Thus researchers came up with ML-based routing techniques. Among the Machine Learning techniques, Reinforcement Learning (RL) proved as a better decision-making approach in the field of computer network traffic control. RL is capable of finding an optimal solution when the system provides different choices. In RL, an agent attempts to learn a policy by continuous interactions with the environment which leads to maximum long-term reward [11]. The reward is a numerical value that is used by the agent to train itself.

A few works based on RL are been reported in the NoC routing domain. Most of them are Q-routing which uses Q-learning techniques [12]. Furthermore, some techniques use trained artificial neural networks focused on traffic prediction [13,14]. Very recently, researchers came up with deep reinforcement learning techniques in NoC for arbitration and fault tolerance [15,16]. While analyzing the results and adaptivity of Q-routing techniques, it seems that as the number of cores increases the Q-table to store the state–action space will be extremely large. As a solution to this, we exploited the use of artificial neural networks to approximate the Q-values for each possible action. In this paper, we would like to develop an optimal adaptive routing algorithm under heavy diverse traffic based on deep reinforcement learning. The major contributions of this paper are summarized as follows:

- We propose an adaptive routing algorithm DeepNR, which improves the performance of NoC systems with a deep reinforcement learning approach. DeepNR tries to observe the state of the router and the long-term performance impact of routing decisions.
- We perform comparison studies with conventional techniques using a real system simulator Gem5 [17]. Simulation results demonstrate that the agent can learn from continuous interaction with the network environment and build an intelligent routing strategy through appropriate reward sets and training.

This paper is organized as follows. Section 2 describes the background of NoC architecture, Reinforcement Learning, and an outline of the related works. In Section 3 we describe the components of the proposed DeepNR and the details of the DeepNR routing algorithm. The performance of DeepNR is evaluated in Section 4. Finally, Section 5 concludes the proposed method.

2. Background

2.1. NoC architecture

The NoC architecture acts as a communication framework for chip based multi-core processors like Intel Xeon-Phi [18] and Tilera TILEPro [19]. Fig. 1 shows a conceptual view of the NoC based Tiled Chip Multicore Processor (TCMP) system. In NoC, tiles are organized in different topologies like mesh, torus, ring, etc, and are interconnected by the routers using bidirectional links, form an underlying NoC (refer Fig. 1(a)). Each tile consists of an out-of-order superscalar processor (core), a private L1 cache, and a slice of shared L2 cache as shown in Fig. 1(b) [20]. The L2 cache is equally sliced among all the cores and is inclusive in nature. Typically, every L1 cache miss triggers packet generation in the NoC network. If a processing core requests for a data element that is not present in the corresponding core's L1 cache, then a miss request is forwarded as a packet to the shared L2 cache block where the data resides. These request packets, data block replies, and coherence packets account for the NoC network traffic. All cache miss requests and replies are carried to the destination tile as per the SNUCA bank mapping policy [21]. A packet travels from source to destination tile by a series of processes like switching, arbitration, and routing through a specific number of cycles. Upon receiving the packet, the destination tile generates a cache block reply packet which reaches the source by similar steps. To better utilize the bandwidth, these packets are further divided into smaller multiple units called flits, in which each flit size is equal to the inter-router link bandwidth. Thus a packet consists of a head flit, body flits, and a tail flit. Usually, the routing is done for the head flit which is followed by body flits and followed by tail flit [22].

In a 2D mesh topology, each NoC router is associated with four neighboring routers in its East, West, North, and South directions as shown in Fig. 1(c). Hence the structure of a baseline NoC router has 5 input and output ports, among them, 4 are connected to neighbors and the remaining one is the local port. Every input port is associated with a set of interim storage space (buffers) called virtual channels to store incoming packets [23]. The use of virtual channels reduces the network latency at the expense of area and power consumption [24]. The packets entering into the input port are routed to the one-hop neighboring router, which is nearest to the packet's destination. Thus a typical baseline NoC router is associated with a routing unit, a virtual channel allocator unit, a switch allocator unit, and a 5 × 5 crossbar (refer Fig. 1(c)). The control information embedded in the packet header helps the routers to determine the proper output port. Once route computation and virtual channel allocation are over, each packet will undergo an arbitration step. During the arbitration, packets competing for the same output ports are wisely selected. A switch allocation is performed after this arbitration, where the packets are assigned to output ports through a crossbar switch.

2.2. Reinforcement learning

Reinforcement Learning (RL) can be modeled as a Markov Decision Process, having a discrete set of states $\mathcal{S}$, a discrete set of actions $\mathcal{A}$, a reward function $\mathcal{F}$ and a state transition function Φ [25]. In RL, the agent interacts with the environment E at discrete time intervals t, where $t \geq 0$. During each time interval, the agent tries to choose the best action a_t from $\mathcal{A}$ observing the current state s_t of $\mathcal{S}$. Upon performing action a_t, the agent obtains a reward $\mathfrak{f}_t$ in response to the transition $\Phi(s_t, a_t)$ and enters into a new state s_{t+1}. This process continues until it reaches a terminal state. The goal of agent is to maximize the long-term reward $\mathcal{F}$, which is the expected cumulative sum of all future rewards $(\mathfrak{f}_t, \mathfrak{f}_{t+1}, \ldots)$

$$\mathcal{F} = \sum_{t=0}^{\infty} \gamma^t \cdot \mathfrak{f}_{t+1}, 0 \leq \gamma \leq 1 \tag{1}$$

In Eq. (1), γ is the discount factor which determines the impact of future rewards by learning a policy. A policy function, denoted by $\pi : \mathcal{S} \rightarrow \mathcal{A}$, explains the set of actions an agent should take for every possible state $s \in \mathcal{S}$. The reward depends on the quality of actions. Since many actions are possible from a specific state, it is important to determine the quality of each action. To determine the quality, each state–action pair is assigned with a numerical value called Q-value denoted by $Q(\mathcal{S}, \mathcal{A})$. Q-values are iteratively updated to find an optimal one and this learning process is called Q-learning.

2.2.1. Q-learning

Q-learning is a value-based RL algorithm, aims to learn the value of an action (quality of an action) in a particular state [26]. It uses a table of Q-values for every state–action pair. Mostly the Q-values are initialized to zero. In each time slot t, Q-learning algorithm takes the action a_t with highest Q-value. After taking an action a_t, $Q(\mathcal{S}, \mathcal{A})$ is updated iteratively using the Bellman's equation:

$$Q_{(s_t,a_t)} = (1-\alpha)Q_{(s_t,a_t)} + \alpha(\mathfrak{f}_t + \gamma_{max_{a_t}} Q_{(s_{t+1},a_{t+1})}) \tag{2}$$

In Eq. (2), α is the learning rate. An appropriate value of α in Q-learning algorithm converges Q-value function to an optimal value Q^* and obtain an optimal policy π^*. The optimal Q-value function Q^* maximizes the expected return [25].

The ϵ-greedy policy is applied to π^* for exploring the unvisited regions of the state–action pair. Instead of always taking the action with the maximum Q-value, here the agent selects a random action with probability ϵ [27].

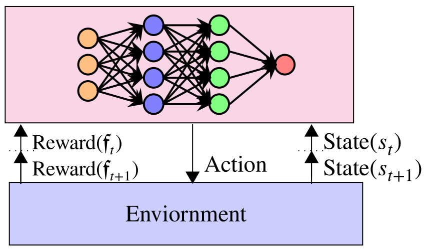

Fig. 2. Deep Reinforcement Learning.

2.2.1.1. Q-Routing.

Q-routing is based on Q-learning, in which a routing policy such as minimizing the packet delivery time is learned. It is an adaptive routing algorithm, in which a node makes the routing decisions based on the neighboring nodes' information [28]. Each node has a Q-table which stores the Q-value which is used to determine the quality of available paths. These values are updated each time a node sends a packet to one of its neighboring nodes. The optimal policy π^* is recorded in the Q-table. When the RL agent observes any new state–action pair, it creates a new entry in the Q-table to record its actions and associated Q-values.

2.2.1.2. Deep Q-routing.

Each state vector is associated with a number of features. As the number of states increases, the memory required to save and update the table details will also increase. Thus the amount of time required to explore each state to create the required Q-table would be impractical. To address this problem, deep Q-learning [29] is introduced, and hence deep Q-routing, in which the state–action table is replaced with an Artificial Neural Network (ANN) as given in Fig. 2. The ANN calculates the state–action value instead of storing the entire state–action table in the router, which in turn eliminates the storage space for state–action pairs.

In reinforcement learning, the initial step is to define the states, actions, and reward for attaining the desired goal. Such requirements of the proposed work are defined in Section 3.2

2.3. Related works

A significant amount of research works have been done in the area of NoC routing. However, machine learning based NoC routing techniques have recently been adopted. Such algorithms employ Q-learning techniques which is RL based one. Q-routing have been initially employed in congested NoC to enhance different levels of Quality-of-Service (QoS) such as best effort and guaranteed throughput [30]. A topology-agnostic fault tolerant deflection routing based on Q-learning to improve NoC performance is also presented [31]. Q-learning based Congestion-aware Algorithm (QCA) presented by Farahnakian et al. alleviates congestion by propagating learning packets with local and non-local information over the network [32]. Farahnakian et al. also proposed a congestion aware Dual Reinforcement Q-routing method (DuQAR), which provides a routing policy to reduce network congestion by estimating latency values between each pair of source and destination nodes in the network [33].

In order to reduce the table overhead incurred by traditional Q-routing, the NoC network is divided into clusters and each cluster maintains intra and inter cluster tables [34]. [35,36] extends this into a hierarchical cluster based Q-routing by removing virtual channels.

In [11], a routing algorithm is selected from a number of standard routing algorithms based on the traffic demand. They developed a RL framework for selecting the best routing algorithm from multiple available routing algorithms. Reza et al. [37] proposed a similar routing technique which is capable of selecting a routing algorithm from three

RL algorithms at runtime depending on the NoC traffic and congestion information.

Considering the above works the presence of Q-tables make it difficult to manage and maintain larger Q-tables as the number of cores in NoC becomes more in future. Thus, instead of using a Q-table to store the entire state–action values, an ANN can be used to find the state–action pair values thereby eliminating the need for large storage space for state–action values. Recently, deep reinforcement learning techniques have been applied to NoC arbitration [15], fault tolerance [16], in routerless NoC architecture [38] and to optimize energy and power [39]. These work inspired us to use the deep reinforcement technique in the design of NoC routing. Thus, to reduce the overhead created by these tables and to increase the routing efficiency by adaptively selecting the low congested minimal paths, we propose an intelligent routing based on deep reinforcement learning for approximating Q-function by eliminating Q-tables. To the extent of our knowledge, this is the first work that uses a deep reinforcement routing method for router-based NoC systems.

3. Methods and design

3.1. System model and problem formulation

We consider the NoC network in $N \times N$ $2D$ mesh topology with N processing cores in all rows and columns. Routers are attached to all cores. Assume the mesh topology as a graph $G(V, E)$ where $V = \{R_1, R_2, \dots, R_N\}$ is the set of N routers associated with the processing cores $P_1, P_2, \dots, P_N$ respectively and $E \subseteq V \times V$ is the set of bidirectional links between each pair of routers. Let $e_{i,j}$ be the link connecting the routers R_i and R_j such that $e_{i,j} \in E$. The neighboring routers of router R_i is denoted by N_{R_i}, where $N_{R_i} = \{R_j \in V | e_{i,j} \in E\}$. A route $\mathcal{H}$ is a sequence of routers $R_{src} \rightarrow R_{j_1} \rightarrow R_{j_2} \rightarrow R_{j_3} \rightarrow \cdots \rightarrow R_{j_n} \rightarrow R_{dest}$ such that packets are traversed from a source router R_{src} to a destination router R_{dest} via the sequence of routers $R_{j_1} \rightarrow R_{j_2} \rightarrow R_{j_3} \rightarrow \cdots \rightarrow R_{j_n}$. The source R_{src} generates a packet P at time t_1, which reaches at R_{dest} in time t_{dest_p} (latency of packet p).

Thus t_{dest_p} is defined as:

$$t_{dest_p} = \sum_{\forall R_i \in \mathcal{H}} t_{R_i} + h \cdot t_l \tag{3}$$

In Eq. (3) t_{R_i} is the time spend by the packet P in router R_i, h is the number of hops through which packet P is traveled to reach R_{dest} and t_l is the time taken to traverse through a link l. Here the performance greatly influenced by the average latency of packets T_{avg},

$$T_{avg} = \frac{\sum_{i=1}^{n} t_{dest_i}}{n} \tag{4}$$

where n is the number of packets in the NoC network.

In the above scenario, we design a new routing algorithm for improving NoC system performance. This objective can be formally stated as: Given $G(V, E)$, R_{src}, and R_{dest} where, $G(V, E)$ is an NoC network in $N \times N$ $2D$ mesh topology, R_{src} is the source router and R_{dest} is the destination router. The goal is to find the shortest path for forwarding packets from R_{src} to R_{dest} using a congestion control load balancing adaptive routing algorithm which minimizes average latency T_{avg}, system routing overhead, and enhances the life of the network.

3.2. DeepNR: Proposed deep reinforcement learning based routing

This section describes our proposed single-agent deep reinforcement-based Q-routing algorithm named DeepNR. Here the routing policy is to minimize the overall packet latency of the entire NoC system. The algorithm uses ϵ-greedy policy and experience replay [40] to adaptively select routing paths.

The DeepNR model is defined as $\langle \mathcal{S}, \mathcal{A}, \mathcal{F}, \Phi, \gamma \rangle$ where $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space, $\mathcal{F}$ is the reward function, Φ is the transition function and $\gamma \in [0, 1]$ is the discount rate. The components of the DeepNR model are:

1. **Environment:** The environment is an NoC network, with processing cores interconnected via routers through bidirectional links.
2. **State:** Each state is a five-dimensional feature vector in which a finite set of easy to compute features from packets and routers are considered. These state features are processed by the agent for taking an action, where the action chooses the routing direction. For every routing decision, the state features are different. Consider a ready to route packet P at router R at discrete time instance t. Then, the state space is $\mathcal{S} = [f_1, f_2, f_3, f_4, f_5]$ where,
    - f_1: Current router id. This is the id of the router R where the packet P stay at time t.
    - f_2: Destination router id. This destination id of the packet P is extracted from the header of P which is encapsulated in the packet header at the time of packet generation.
    - f_3: Distance traversed. It is the total distance traversed by P so far. The distance of travel, which is the number of hops P traversed, influences the latency of a traffic flow. Hence, the distance is added as a field in the packet at the time of generation. This value will be incremented by one during each hop traversal.
    - f_4: Remaining distance, the minimum number of hops the packet has to travel from the current router to reach the destination router. It is calculated as the difference between the total distance ($total_d$) from source to destination router and the distance already traversed by the packet ($traversed_d$). Each router in the NoC network has assumed to have coordinates (x,y) and an approximate value of the $total_d$ is calculated based on the coordinate values. Assume that the current and destination routers have coordinates (c_x, c_y) and (d_x, d_y) respectively. Then the remaining distance ($remaining_d$) is calculated as,

      $$remaining_d = total_d - traversed_d \tag{5}$$

      where $\; total_d = (\left|c_x - d_x\right| + \left|c_y - d_y\right|)$.
    - f_5: Free buffer vector. It is a four-dimensional vector, $B_t = (B_t^1, B_t^2, B_t^3, B_t^4)$, which represents the number of free buffer slots available in the neighboring routers of R at time t. Modern TCMPs are input buffered, hence they employ a minimum number of buffers in each router for bandwidth scalability. Thus, the packets coming from neighboring routers are initially stored in the input buffers of the router. If there is no free buffer available, it implies that the neighboring router is experiencing a processing delay which may lead to local congestion thereby leads to global congestion. Hence buffer slots are used as a measure of congestion.

   Each of the features discussed above is normalized, such that all state vector features fall in the same range of values.
3. **Action space:** It is a four-dimensional vector, $\mathcal{A} = [a_1, a_2, a_3, a_4]$, which represents the four routing directions possible for a router in a 2D mesh NoC environment. Here, the action a_t selects the next hop router for a packet P which is ready to be routed in the input buffer of R at time t. The Q-values generated by the agent correspond to these actions. For the routers present in the mesh borders, the routing directions are restricted to less than four, hence these moves are treated as invalid actions in DeepNR, and the NoC environment penalizes these actions by giving negative rewards. Thus the agent learns to maximize reward by not executing any invalid actions.
4. **Reward:** Rewards play an important role in RL, as training with wrong rewards will result in improper optimization of the network. The NoC environment generates and sends a reward to the agent at time $t + 1$, after performing the action selected by the agent at time t. This reward decides how effective is the

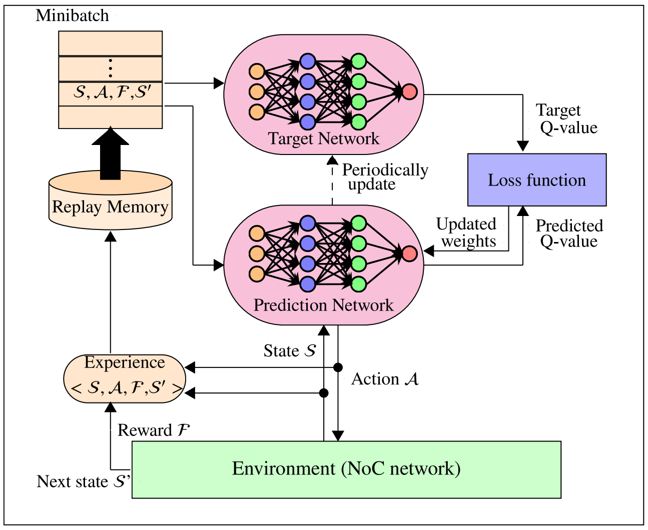

Fig. 3. Proposed DeepNR model.

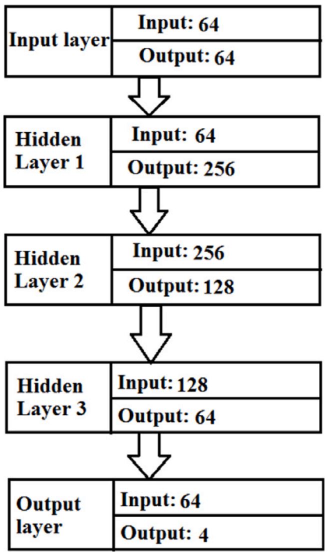

Fig. 4. Neural network model.

action performed in the current environment state. Since the objective of the proposed algorithm is to reduce packet latency, the following rewards are considered:

- Queuing delay: It is the amount of time a packet resides in the router buffer until it gets routed to the neighboring router. The queuing delay of packets is calculated for every action taken and the reciprocal of which is used in reward calculation to avoid larger values.
- Constant reward: A fixed value reward is used for each action. Sometimes the network tends to take invalid routing decisions, such as selecting routing directions for routers placed in mesh border, resulting in unresolved conditions. To resolve this, a new state called the *'dead state'* is defined to track the invalid actions taken. If the packet reaches *'dead state'* because of the invalid decisions, a constant negative reward is given to the neural network agent. Thus, the agent is penalized heavily when it chooses a *'dead state'*.

In this work, we designed a neural network which replaces the standard Q-tables in Q-learning to provide better results by making the model more versatile. Here, the agent interacts with the environment (NoC network), at discrete intervals of time t. At t, the agent observes and gathers the state features of the environment and selects an action from the four discrete actions available. Specifically, the input to the neural network are the attribute values of state space and the outputs are the Q-values. Upon performing an action, the environment moves to the next state s_{t+1} with new features and at the same time, the agent receives a reward. The agent receives the new state features in the next time slot and the process iterates over. There is a policy used for mapping from states to action, which specifies how to choose actions with the given state features to maximize cumulative reward. The DeepNR algorithm formulates the policy, based on the agent's past interactions with the NoC system. A conceptual view of the proposed DeepNR model is given in Fig. 3.

3.2.1. Architecture of proposed neural network

The neural network used for training the model is a multi-layer feed-forward perceptron whose architecture is shown in Fig. 4. It consists of three hidden layers with a decreasing number of neurons at each layer. The activation functions selected, for the hidden and output layer neurons, ensure that the network returns Q-values instead of the probability of the action to be taken. The activation functions used are Relu activation and linear activation for hidden layers and output layer respectively, whereas the loss function used is the Mean Squared Error (MSE) with the optimizer as Adam. During training two deep neural networks are used in the algorithm: a prediction network and a target network. The prediction network is used for delivering the action that the router needs to perform. The target network is used to compute the target (ground-truth) Q-values that are used in the training process for computing the Bellman equation and updating the parameters.

In DeepNR (Fig. 3), whenever a router needs to make a routing decision, the current state information is passed to the agent (neural network) using separate control links. Hence, we assumed the latency for transferring this information to be zero. The agent learns the status information of the environment (NoC network) and predicts the future actions to be taken by the router.

We use the ϵ-Greedy approach to slowly explore the environment (NoC network) by taking random actions. After performing the action (routing), the environment sends a reward and next state features to the agent. Each routing experience is represented as a quadruple $\langle \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{S}_{new} \rangle$, meaning an execution of action $\mathcal{A}$ in a state $\mathcal{S}$ results in new state $\mathcal{S}_{new}$ and reward $\mathcal{F}$. To improve the stability of the network, it is better to reuse past experiences instead of the latest ones. Thus, these experiences are stored in a memory buffer called experience replay memory, where it has limited entries and the oldest record is overwritten by a new record. Since the neural networks are more efficient when they are trained using batches of experiences, experience replay records are randomly sampled to form batches while training the model. As the routing of a packet involves a sequence of decisions, iterative sampling helps the network to identify patterns between routing decisions of the same packet. Since the dataset is sampled randomly from experience replay, it ensures the disparity of subsequent training points which in turn reduces the chances of the neural network falls into local minima. In addition, experience replay allows the model to learn from the past data points multiple times which leads to a faster model convergence.

Among the two neural networks used, the prediction network generates Q-values for making routing decisions whereas the target network generates Q-values that are used to update neural network parameters. During training, each data element from a batch is passed to these two networks. The prediction network takes the current state and action as input and outputs the predicted Q-values for that particular transition.

Meanwhile, the target network takes the next state from that sample as input and predicts the Q-value which acts as the future Q-value. The Mean Squared Error (MSE) between the target and predicted Q-values are used to compute the loss to train the prediction network. In order to ensure that the Q-values remain stable for a short period of time, the target network is trained only periodically. The weights of the prediction model are updated by performing the gradient descent after computing the loss, whereas the weights of the target model are updated periodically. This process continues during the course of the algorithm.

3.2.2. Learning algorithm

A detailed learning algorithm of the DeepNR model is illustrated in Algorithm 1. The input to the algorithm is the state information and the output is the state–action value function. In an NoC network, if there are packets waiting in routers which are ready to be routed then the intelligent routers map the current environment information to a unique representation called *state* which is fed as input to the agent. Among the state features, the *router IDs* (feature f_1 and feature f_2) and *distance traversed* (feature f_3) are extracted from the incoming packets. The *remaining distance* (feature f_4) is as in Eq. (5). The free buffer/virtual channel information of each router needs to be communicated with its neighboring routers, where the buffer/virtual channel allocation and deallocation is done by the VC allocator. After extracting all these 5 features, these are fed as input to the proposed algorithm.

At the beginning of the algorithm, the prediction and target Q networks are initialized with weights w and $\hat{w}$ respectively. The experience replay memory is also initialized in this step. The agent performs the following steps on each time slot from 1 to T: it observes the state and chooses an action a_t using ϵ-greedy policy. The agent can select either a random action a_t with ϵ probability, or the action having a maximum Q-value with 1-ϵ probability.

$$a_t = \begin{cases} \text{random action,} & \text{probability } \epsilon \\ argmax_a Q_{w_t}(s_t, a_t), & \text{probability 1-}\epsilon \end{cases} \tag{6}$$

After selecting the route (action) corresponding packet is forwarded to the particular direction. The action is executed and the NoC environment calculates the reward and sends it back to the agent. Meanwhile, it also set a flag d_t, indicating whether the packet has reached its destination where,

$$d_t = \begin{cases} 1, & \text{if packet reaches destination} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

The current interaction tuple is stored in the replay memory M and the training of prediction and target networks are performed accordingly.

4. Results and discussions

This section describes the simulation setup and the performance analysis of the proposed model.

4.1. Simulation setup

We use the Gem5 simulator which is a full system simulator having a Garnet module for the evaluation. The Garnet module simulates NoC cycles accurately. The proposed DeepNR is integrated into the Garnet module of the Gem5 simulator. For the simulation, an NoC with 64 cores arranged in 8 × 8 mesh topology is considered. Each router is set up with two virtual channels which are having four flit buffers. The request packets are assumed to be one flit packet whereas the response packets are 4 flits in length.

As mentioned earlier, the neural network of the DeepNR model consists of an input layer, 3 hidden layers, and an output layer. The DeepNR parameters are the discount factor (γ), learning rate (α), and ϵ. The discount factor (γ) determines how much the reinforcement

Algorithm 1: Proposed DeepNR algorithm

```
   Input: The state information $\mathcal{S} = [f_1, f_2, f_3, f_4, f_5]$
   Output: State–action value function, $Q_w(s)$
 1 Initialize the prediction network $Q$ and the target network $\hat{Q}$
     with random weights $w$ and $\hat{w}$ respectively, where $w = \hat{w}$.
 2 Initialize the experience replay memory $M$
 3 Initialize state $s_1 = [f_1^1, f_2^1, f_3^1, f_4^1, f_5^1]$
 4 for t=1 to T do
 5     Observe the state $s_t$
 6     Choose an action $a_t$ using policy or $\epsilon$ - greedy approach,
           $a_t = \begin{cases} \text{random action,} & \text{probability } \epsilon \\ argmax_a Q_{w_t}(s_t), & \text{probability 1-}\epsilon \end{cases}$
 7     Execute $a_t$ by forwarding packet to next router
 8     Set the flag $d_t$,
           $d_t = \begin{cases} 1, & \text{if packet reaches destination} \\ 0, & \text{otherwise} \end{cases}$
 9     Obtain the reward $\mathfrak{r}_t$ and next state $s_{t+1}$ generated by the
         NoC network.
10     Store interaction tuple $(s_t, a_t, s_{t+1}, \mathfrak{r}_t, d_t)$ in $M$
11     for every $T_{train}$ do
12         if enough experience in $M$ then
13             Randomly sample the transitions $(s_i, a_i, s_{i+1}, \mathfrak{r}_i, d_i)$
               from $M$ and form a mini-batch $M_s$
14             for each transition $(s_i, a_i, s_{i+1}, \mathfrak{r}_i, d_i)$ in $M_s$ do
15                 if $d_i$ then
16                     Compute the target value $y_i = \mathfrak{r}_i$
17                 else
18                     Compute the target value $y_i =$
                         $\mathfrak{r}_i + \gamma max_{a_{i+1}} \hat{Q}(s_{i+1}, a_{i+1}; \hat{w})$
19         Calculate loss using mean square error,
               $L(w) = \frac{1}{M} \sum_M (Q_w(s_i, a_i) - y_i)^2$
20             Update $w$, weights of prediction network $Q$, by
               performing gradient descent by minimizing the loss.
21     for every $T_{target}$ do
22             Copy the weights of prediction network $Q$ to target
               network $\hat{Q}$, $\hat{w}$=$w$
23 return the $Q$-value function from prediction network $Q_w$
```

Table 1
Classification of SPEC CPU 2006 benchmark applications.

Miss rate in MPKI	Benchmarks
Low	gromacs, calculix, h264ref, gobmk,
Medium	bwaves, gcc, gamess, bzip2
High	lbm, leslie3d, mcf, hmmer

learning agents care about rewards in the distant future relative to those in the near future, the learning rate (α) explicitly defines how much we are updating the Q-value, and ϵ defines how random action is taken. We perform sensitivity analysis based on Eq. (1), Eq. (2), and Eq. (6) for the values from 0 to 1 and find appropriate values that work better for the DeepNR model. From Eq. (1) it is observed that, when γ = 0, the learning is nearsighted, hence it does not consider future rewards instead immediate rewards are considered. When γ = 1, more weightage is given to future rewards other than the immediate reward. This causes too early convergence which results in failure to converge to an optimal solution. Thus, by analysis it is found that 0.9 for γ achieves better performance. While considering the value of α,

Table 2
Workload Mixes with different network injection intensity.

Workloads	SPEC CPU 2006 benchmark applications			
W1	gromacs (16 cores)	calculix (16 cores)	h264ref (16 cores)	gobmk (16 cores)
W2	gromacs (16 cores)	calculix (16 cores)	bwaves (16 cores)	gcc (16 cores)
W3	bwaves (16 cores)	gcc (16 cores)	bzip2 (16 cores)	gamess (16 cores)
W4	gromacs (16 cores)	calculix (16 cores)	lbm (16 cores)	leslie3d (16 cores)
W5	lbm (16 cores)	leslie3d (16 cores)	bwaves (16 cores)	bzip2 (16 cores)
W6	lbm (16 cores)	leslie3d (16 cores)	mcf (16 cores)	hmmer (16 cores)

in Eq. (2), $\alpha = 0$ shows the Q-values never get updated hence old values are reused, and when we choose higher values for α it leads to inaccurate convergence of Q-values. Thus, we fixed 0.01 as the learning rate since accuracy is more important than speed. Now, consider the value of ϵ, from Eq. (6), it is clear that the lower value of ϵ results in a pure greedy approach while the higher value selects too random routing decisions. To explore and learn, it is better to make random decisions at the beginning of the learning process, and as the learning progresses, lower values of ϵ perform better. Therefore, initially, ϵ is set to the value 0.9 and, as the learning progress, it is decayed to 0.01, so as to stabilize and exploit the policy. We use both synthetic traffic and SPEC 2006 multi-programmed CPU benchmark applications [41] to evaluate the performance of the DeepNR model.

4.1.1. *Synthetic traffic*

For performance analysis, four standard synthetic traffic patterns, namely uniform, transpose, bit complement, and shuffle, are used. In uniform traffic, the destination is selected for each generated packet according to a uniform distribution. For transpose traffic, the destination of the packet generated from the source (x,y) is always (y,x). The packet's destination is always the one's complement of the packet's source address in bit complement traffic. In shuffle traffic, the packet's destination is always the left circular shift of the packet's source address. Here, the network performance metrics are evaluated by varying injection rates from zero to saturation point.

4.1.2. *Real traffic*

The performance of our proposed model is also compared with SPEC 2006 CPU benchmarks, the real traffic loads. For analysis SPEC 2006 CPU benchmarks are classified based on Misses Per Kilo Instructions (MPKI). The applications which are having less than 5 misses per 1000 instructions are grouped into low MPKI applications as shown in Table 1. Similarly, for medium MPKI applications misses are between 5 and 25, and greater than 25 will fall under high MPKI applications. Six random workload mixes of these applications are used. Each workload mix is classified into different network injection intensity groups and every mix is a combination of four applications which is discussed in Table 1.

Six workload mixes (W1–W6) were created using SPEC benchmark applications which are having different intensity groups. These workload mixes are arranged in the increasing order of their network injection intensities in Table 2, where W1 have the least intensity and W6 have the highest. Consider workload mix 1 (W1): out of the 64 cores, we model for simulation, 16 cores run $gromacs$ application, 16 cores load $calculix$, 16 core engage $h264ref$ and the remaining 16 executes $gobmk$. Other workload mixes are also created in a similar way.

4.2. *Performance analysis*

For analyzing the results, the proposed routing algorithm DeepNR is compared with existing XY-routing [42], 2-way OE [43] and Q-routing algorithms.

Fig. 5 shows the comparison between average packet latency and injection rates using the synthetic traffics — uniform, transpose, bit complement, and shuffle. Average packet latency is defined as the average of the total number of clock cycles every packet spent in the NoC network, whereas the injection rate is defined as the probability with which packets are injected from a core in every cycle.

During lower injection rates the amount of traffic in the network is lower and when the injection rate increases the traffic in the network also increases. Hence the congestion will be more during high injection rates. From the latency curves, it is quite evident that the packet latency is always directly proportional to traffic injection rates in the network. But the interesting fact is that, while comparing the overall packet latency among the conventional routing algorithms, it is clear that DeepNR has the minimum latency. In the initial state of the network with low traffic, DeepNR performs almost similar to all the four routing algorithms. It shows the effectiveness of DeepNR in learning an optimal policy from the earlier stages. As network traffic increases (during congestion), packet latency also increases resulting in a degradation in performance. In the XY routing algorithm, this degradation is due to the fact that the packet always flows through a definite path. If there is congestion in that path, the packets will wait in the router buffer. 2-way OE is a deflection routing algorithm which selects an alternate route in the presence of local congestion. But as the number of packets increases within a limit, the performance of 2-way OE starts degrading. When compared with Q-routing, DeepNR starts performing better as the network traffic increases from medium to high. This shows the efficacy of DeepNR model in handling adaptive routing decisions even in high network traffic by managing congestion situations.

Another important fact that can be inferred from Fig. 5 is regarding the latency with which different models approach the saturation point. The saturation point is defined as the point where the overall latency of packets reaches twice the minimum delay. It is seen that before reaching saturation point the latency maintains a slow trend in increase. But when the injection rate goes beyond saturation point, the latency will dramatically increase. This means that at saturation point the NoC packets are blocked that increase the latency of the packets results with decrease in the system performance. Therefore it is better to reach the saturation point slowly. From Fig. 5, it can be noticed that the DeepNR model is approaching the saturation point at a later point of time. This means that DeepNR model can afford the highest load among others.

Fig. 6 shows the simulation results of throughput percentage attained with increasing injection rate in the four algorithms used for comparison. Similar to packet latency, it is clear from the graph that the throughput always drops when the injection rate reaches the saturation point. The best value of throughput is one, however this is possible only if packets are injected into the network in a lower rate. Thus, the average value of throughput lowers as the network traffic increases.

Fig. 7 characterize the average packet latency of the entire NoC system for real workloads. Among the workloads, W1 has the minimum traffic and W6 has the maximum. Hence, the packets in W1 experience low latency, and W6 has the highest latency. The graph implies that the system performance of the DeepNR model is improved even in the high network intensity applications' workload.

Fig. 8 compares the throughput percentage attained for SPEC CPU 2006 workloads. It can be seen that the throughput is lowering, from W1 to W6, for all routing techniques due to the increased traffic in the network. Moreover, the DeepNR is always attaining more than 90% in high network traffic conditions.

Fig. 9 shows a comparative study of average hop count traveled by NoC packets under real application workloads. In all cases, XY routing

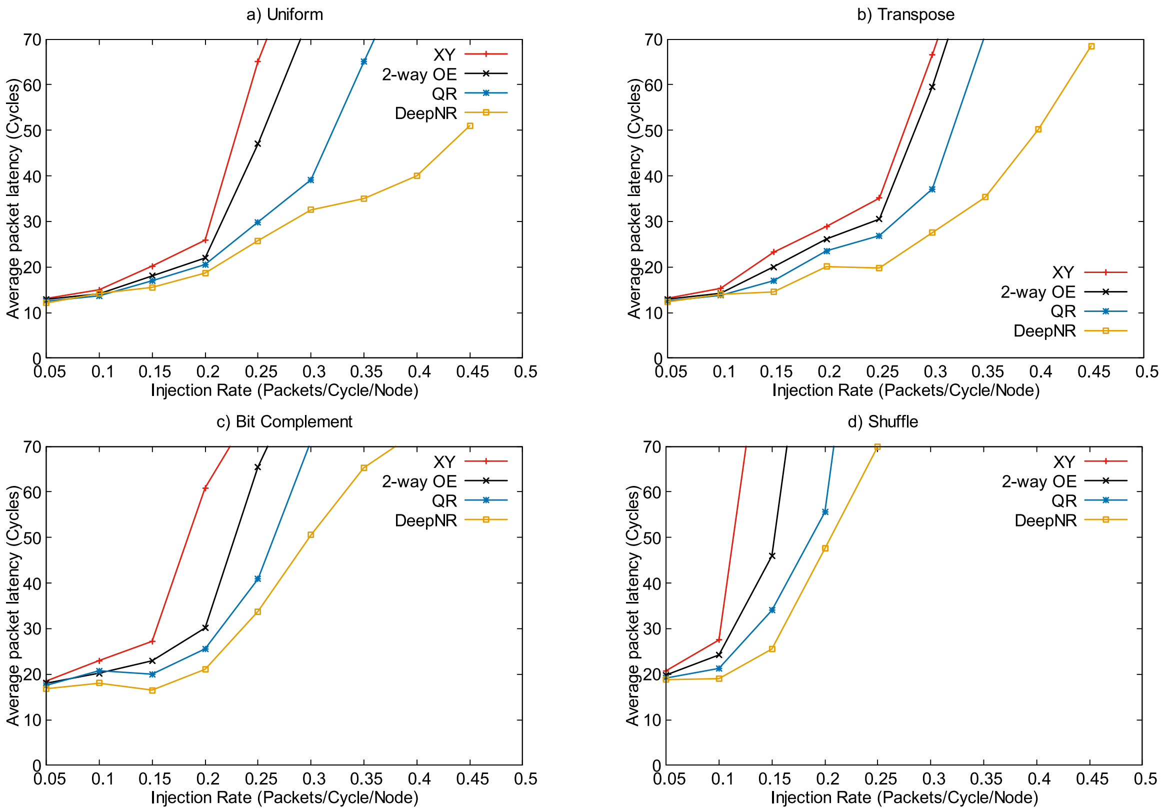

Fig. 5. Average packet latency comparisons for the synthetic traffics: (a) Uniform, (b) Transpose (c) Bit Complement (d) Shuffle.

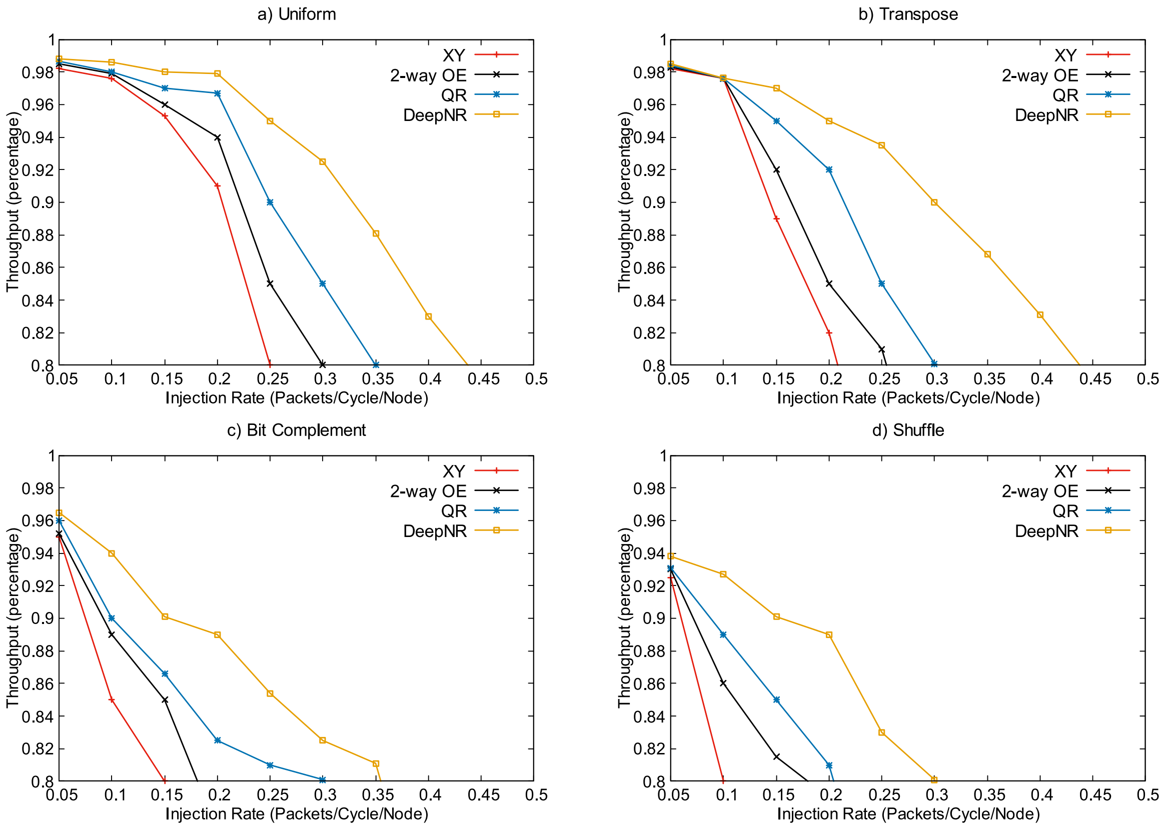

Fig. 6. Percentage (%) throughput comparisons for the synthetic traffics: (a) Uniform, (b) Transpose (c) Bit Complement (d) Shuffle.

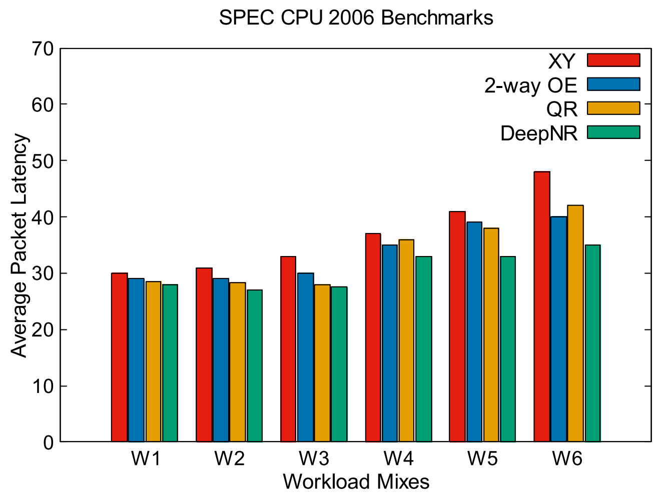

Fig. 7. Average packet latency comparison for real applications.

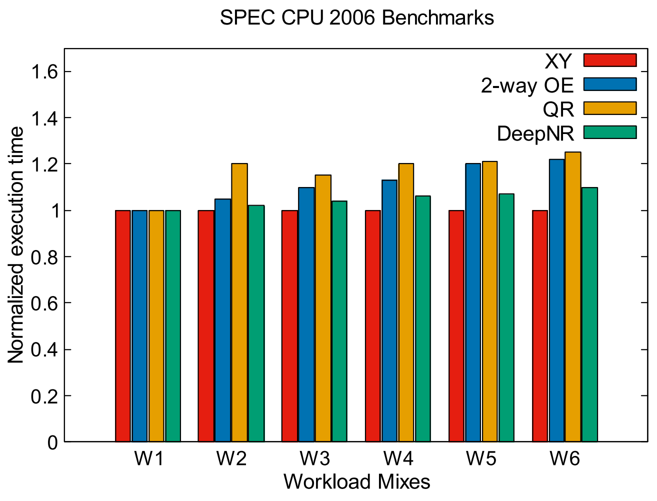

Fig. 10. Execution time comparison for real applications.

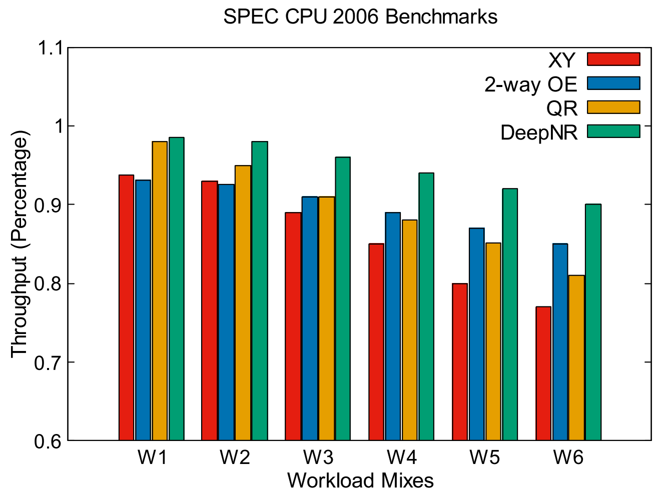

Fig. 8. Percentage (%) throughput comparison for real applications.

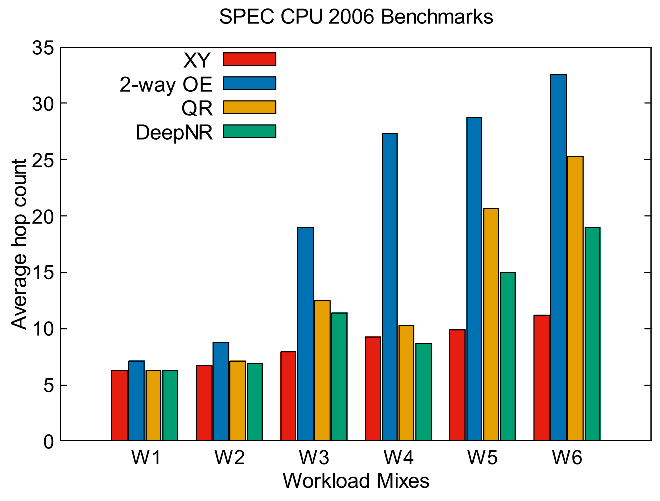

Fig. 9. Average hop count comparison for real applications.

is having fewer hops as it always follows the shortest path. But as congestion increases, the packets get stagnated in the routers which results in the degradation of throughput. It is clear from Fig. 8 that the throughput attained by XY routing is very less. Since 2-way OE routing is a type of deflection routing, it has high hop counts when congestion in the network increases. While comparing with Q-routing, DeepNR is having a low hop count which shows its capability of choosing minimal low congested paths even during high traffic.

In Fig. 10, the proposed DeepNR in SPEC 2006 application environment is compared for execution time and the results are normalized to XY routing. Under less congested workload W1, all the four routing algorithms take almost similar execution times. For DeepNR, the execution time overhead includes the time taken by neural networks to calculate Q-values. Thus, from medium to high congested workloads, the time taken by DeepNR is slightly increased. But even in a highly congested workload, W6 DeepNR performs favorably well.

4.3. Overhead analysis

Timing overhead: The timing overhead of DeepNR routing comprises of the computation time required for calculating action values by the neural network agent, and traversing and updating the state features. The complexity of action calculation depends on the structure of the neural network. For each layer of a neural network, a matrix multiplication and an activation function is applied. Let $U_1, U_2, \dots, U_{\mathcal{L}}$ be the $\mathcal{L}$ layers in the neural network and the maximum number of neurons in a layer U_i is $\mathcal{N}$. Suppose $\mathcal{N}_{mul}$ is the total number of multiplications needed and $\mathcal{N}_{act}$ is the total number of times activation function has been applied. Then, $\mathcal{N}_{mul}$ is $\mathcal{L}$ times the number of multiplications needed in each layer. Since the naive matrix multiplication has an asymptotic run-time of $\mathcal{O}(\mathcal{N}^3)$, the time complexity of $\mathcal{N}_{mul}$ is $\mathcal{O}(\mathcal{L}\mathcal{N}^3)$. Similarly, $\mathcal{N}_{act}$ is $\mathcal{L}$ times the number of activations applied in each layer. Since activation function is an element-wise function, it has a run-time of $\mathcal{O}(\mathcal{N})$. Thus, the total run-time of $\mathcal{N}_{act}$ is $\mathcal{O}(\mathcal{L}\mathcal{N})$. This leads to a total run-time of $\mathcal{O}(\mathcal{L}\mathcal{N}^3)$ for action calculation. Coming to the traversal and updation of feature values, only a constant time is required as dedicated links and modules are used here.

Area overhead: The area overhead of DeepNR model is calculated using the DSENT tool [44] with 45 nm processor technology. Comparisons have been made regarding the area overhead of DeepNR model with that of other XY-routing and Q-routing models. The DeepNR requires an additional circuitry for feature extraction, neural network agent and experience replay. In order to maintain data correlation in experience replay memory, older data is overwritten by new data values. Thus, it requires only a limited number of entries (in DeepNR, 200 entries), which makes the area overhead of experience replay memory insignificant. The state features such as router ids and buffer

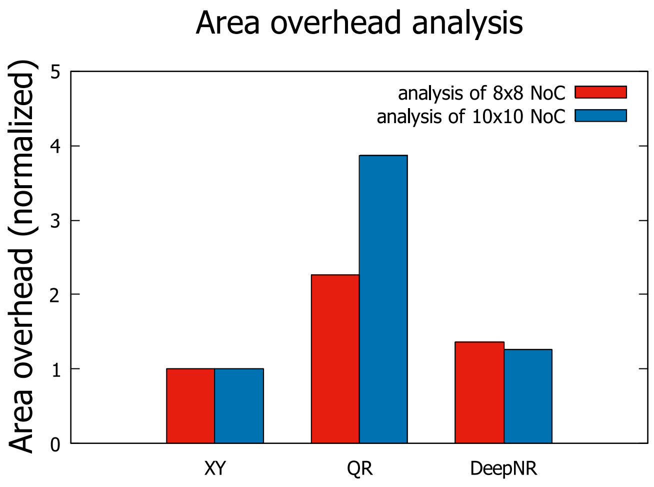

Fig. 11. Normalized area overhead comparisons of DeepNR with XY and Q-routing.

slots can be directly computed from the packet to be routed, and from the routers where the packet is currently residing. To track the distance covered by each packet, an additional 4-bit field is used in the header of each packet. Then, the area overhead of the agent is due to the multipliers used for multiplication and addition operations during the action calculation.

Fig. 11 shows an area overhead analysis of XY, Q-routing and DeepNR models normalized to XY model using 8×8 and 10×10 NoC network. From evaluation, for 8×8, the proposed neural network agent requires an overall area of $0.2158\ \mathrm{mm}^2$, whereas the DeepNR model requires an overall area of $0.4521\ \mathrm{mm}^2$. It can be observed that the DeepNR model shows a reduction of 66% in area as compared to Q-routing, and only a negligible increase of 5.78% in area overhead with XY routing. Further, an area overhead analysis on a 10×10 NoC network (refer Fig. 11) shows a reduction in area by 71% and an increase in area of 4.1% for DeepNR as compared to Q-routing and XY routing respectively. This marginal increase in area is due to the extra space needed for the central agent and additional modules in the intelligent routers.

In this work, a centralized deep reinforcement-based method is implemented, which can continually learn the changing traffic behavior of real-time applications. We used an offline trained neural network agent and integrated it into DeepNR model. Based on the overhead analysis of DeepNR on higher dimensional networks, it is identified that offline trained agent which is integrated into DeepNR model does not incur much overhead to the NoC system, instead, it gives better performance. It creates only a comparable area overhead as that of XY routing with dedicated big matrix multipliers, at the same time providing considerable reductions of up to 21.25% and 44% in packet latency under real and synthetic traffic respectively. In real hardware, a small dedicated neural processing unit for the central agent, which is built using ASIC, can be used. For large scale NoCs, the overhead caused by a single agent will definitely be high. Hence, a more feasible approach is to use distributed multi-neural network agents that support multiple cores in the NoC.

5. Conclusion

The dynamic traffic demands of applications in NoC create network congestion which adversely affects the system performance. Hence, an intelligent routing algorithm is needed to balance the network traffic and thereby mitigate congestion. This paper demonstrates the effectiveness of applying deep reinforcement learning in the design of NoC routing policies. The proposed DeepNR model observes the current network state and predicts the shortest routing path that maximizes the NoC performance. The experimental evaluations show that the DeepNR model is effective in reducing routing overhead which in turn improves the overall system performance.

As part of future work, we are planning to extend the design so as to adapt it to operate in multi-agent environments. This requires careful synchronization of communication among different agents. In addition to that, the proposed framework has applicability in various NoC design problems. Also, it will not be an overstatement to say that such a design can be extended to enable intelligent design space exploration in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] John L. Hennessy, David A. Patterson, Computer Architecture: A Quantitative Approach, fifth ed., Elsevier, 2011.

[2] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, Li-Shiuan Peh, Research challenges for on-chip interconnection networks, IEEE Micro 27 (5) (2007) 96–108, http://dx.doi.org/10.1109/mm.2007.4378787.

[3] Jose Duato, Sudhakar Yalamanchili, Lionel Ni, Interconnection Networks, Elsevier, 2003, http://dx.doi.org/10.1016/b978-1-55860-852-8.x5000-7.

[4] W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), ACM, 2001, http://dx.doi.org/10.1109/dac.2001.935594.

[5] Luca Benini, Giovanni De Micheli, Networks on chips: A new soc paradigm, Computer 35 (1) (2002) 70–78.

[6] International Technology Roadmap for Semiconductors 2015, 2015, https://www.semiconductors.org/resources/2015-international-technology-roadmap-for-semiconductors-itrs/. (Accessed 02 August 2021).

[7] Pradip Kumar Sahu, Santanu Chattopadhyay, A survey on application mapping strategies for network-on-chip design, J. Syst. Archit. 59 (1) (2013) 60–76, http://dx.doi.org/10.1016/j.sysarc.2012.10.004.

[8] Asma Benmessaoud Gabis, Mouloud Koudil, NoC routing protocols – objective-based classification, J. Syst. Archit. 66–67 (2016) 14–32, http://dx.doi.org/10.1016/j.sysarc.2016.04.011.

[9] Khurshid Ahmad, Muhammad Sethi, Review of network on chip routing algorithms, EAI Endor. Trans. Context-Aware Syst. Appl. 7 (22) (2020) 167793, http://dx.doi.org/10.4108/eai.23-12-2020.167793.

[10] Jeffrey Dean, The deep learning revolution and its implications for computer architecture and chip design, in: IEEE International Solid- State Circuits Conference, IEEE, 2020, http://dx.doi.org/10.1109/isscc19947.2020.9063049.

[11] Sheng-Chun Kao, Chao-Han Huck Yang, Pin-Yu Chen, Xiaoli Ma, Tushar Krishna, Reinforcement learning based interconnection routing for adaptive traffic optimization, in: Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip, ACM, 2019, http://dx.doi.org/10.1145/3313231.3352369.

[12] Drew D. Penney, Lizhong Chen, A survey of machine learning applied to computer architecture design, 2019, arXiv preprint arXiv:1909.12373.

[13] Boqian Wang, Zhonghai Lu, Shenggang Chen, ANN Based admission control for on-chip networks, in: Proceedings of the 56th Annual Design Automation Conference 2019, ACM, 2019, http://dx.doi.org/10.1145/3316781.3317772.

[14] Vassos Soteriou, Theocharis Theocharides, Elena Kakoulli, A holistic approach towards intelligent hotspot prevention in network-on-chip-based multicores, IEEE Trans. Comput. 65 (3) (2016) 819–833, http://dx.doi.org/10.1109/tc.2015.2435748.

[15] Jieming Yin, Yasuko Eckert, Shuai Che, Mark Oskin, Gabriel H. Loh, Toward more efficient noc arbitration: A deep reinforcement learning approach, in: Proceedings of the 1st International Workshop on AI-Assisted Design for Architecture, AIDArc, 2018.

[16] Ke Wang, Ahmed Louri, CURE: A High-performance, low-power, and reliable network-on-chip design using reinforcement learning, IEEE Trans. Parallel Distrib. Syst. 31 (9) (2020) 2125–2138, http://dx.doi.org/10.1109/tpds.2020.2986297.

[17] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, David A. Wood, The gem5 simulator, ACM SIGARCH Comput. Architect. News 39 (2) (2011) 1–7, http://dx.doi.org/10.1145/2024716.2024718.

[18] Intel Xeon Phi Processor 7235, https://ark.intel.com/content/www/us/en/ark/products/128694/intelxeon-phi-processor-7235-16gb-1-3-ghz-64-core.html. (Accessed 02 August 2021), 2021.

[19] Tilera processor family, http://www.tilera.com/. (Accessed 02 August 2021), 2021.
[20] Érika Cota, Alexandre de Morais Amory, Marcelo Soares Lubaszewski, NoC basics, in: Reliability, Availability and Serviceability of Networks-on-Chip, Springer US, 2011, pp. 11–24, http://dx.doi.org/10.1007/978-1-4614-0791-1_2.
[21] Changkyu Kim, Doug Burger, Stephen W. Keckler, An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, ACM SIGARCH Comput. Archit. News 30 (5) (2002) 211–222, doi:10.1145%2F635506.605420.
[22] Daniel Sanchez, George Michelogiannakis, Christos Kozyrakis, An analysis of on-chip interconnection networks for large-scale chip multiprocessors, ACM Trans. Archit. Code Optim. 7 (1) (2010) 1–28, http://dx.doi.org/10.1145/1736065.1736069.
[23] John Jose, J. Shiva Shankar, K.V. Mahathi, Damarla Kranthi Kumar, Madhu Mutyam, BOFAR, in: Proceedings of the 4th International Workshop on Network on Chip Architectures, NoCArc '11, ACM Press, 2011, http://dx.doi.org/10.1145/2076501.2076506.
[24] William J. Dally, Virtual-channel flow control, ACM SIGARCH Comput. Archit. News 18 (2SI) (1990) 60–68, http://dx.doi.org/10.1145/325096.325115.
[25] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, second ed., MIT Press, 2018.
[26] Christopher J.C.H. Watkins, Peter Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292, doi:10.1007.bf00992698.
[27] Quintin Fettes, Mark Clark, Razvan Bunescu, Avinash Karanth, Ahmed Louri, Dynamic voltage and frequency scaling in NoCs with supervised and reinforcement learning techniques, Computer 52 (9) (2019) 4–5, http://dx.doi.org/10.1109/mc.2019.2923827.
[28] Justin A. Boyan, Michael L. Littman, Packet routing in dynamically changing networks: A reinforcement learning approach, in: Advances in Neural Information Processing Systems, 1994, pp. 671–678.
[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533, http://dx.doi.org/10.1038/nature14236.
[30] Krishan Kumar Paliwal, Jinesh Shaji George, Navaneeth Rameshan, Vijay Laxmi, M.S. Gaur, Vijay Janyani, R. Narasimhan, Implementation of QoS aware Q-routing algorithm for network-on-chip, in: Communications in Computer and Information Science, Springer, 2009, pp. 370–380, http://dx.doi.org/10.1007/978-3-642-03547-0_35.
[31] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, Minxuan Zhang, A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for Network-on-Chip, in: Proceedings of the Third International Workshop on Network on Chip Architectures, NoCArc '10, ACM Press, 2010, http://dx.doi.org/10.1145/1921249.1921254.
[32] Fahimeh Farahnakian, Masoumeh Ebrahimi, Masoud Daneshtalab, Pasi Liljeberg, Juha Plosila, Q-learning based congestion-aware routing algorithm for on-chip network, in: 2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications, IEEE, 2011, http://dx.doi.org/10.1109/nesea.2011.6144949.
[33] Fahimeh Farahnakian, Masoumeh Ebrahimi, Masoud Daneshtalab, Juha Plosila, Pasi Liljeberg, Adaptive reinforcement learning method for Networks-on-Chip, in: 2012 International Conference on Embedded Computer Systems, SAMOS, IEEE, 2012, http://dx.doi.org/10.1109/samos.2012.6404180.
[34] Manas Kumar Puthal, Virendra Singh, M.S. Gaur, Vijay Laxmi, C-routing: an adaptive hierarchical noc routing methodology, in: 2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip, IEEE, 2011, http://dx.doi.org/10.1109/vlsisoc.2011.6081616.
[35] Masoumeh Ebrahimi, Masoud Daneshtalab, Fahimeh Farahnakian, Juha Plosila, Pasi Liljeberg, Maurizio Palesi, Hannu Tenhunen, HARAQ: COngestion-aware learning model for highly adaptive routing algorithm in on-chip networks, in: 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip, IEEE, 2012, http://dx.doi.org/10.1109/nocs.2012.10.
[36] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, Bi-LCQ: A low-weight clustering-based Q-learning approach for NoCs, Microprocess. Microsyst. 38 (1) (2014) 64–75, http://dx.doi.org/10.1016/j.micpro.2013.11.008.
[37] Md Farhadur Reza, Tung Thanh Le, Reinforcement learning enabled routing for high-performance networks-on-chip, in: 2021 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2021, http://dx.doi.org/10.1109/iscas51556.2021.9401790.
[38] Ting-Ru Lin, Drew Penney, Massoud Pedram, Lizhong Chen, A deep reinforcement learning framework for architectural exploration: A routerless NoC case study, in: IEEE International Symposium on High Performance Computer Architecture, HPCA, IEEE, 2020, http://dx.doi.org/10.1109/hpca47549.2020.00018.
[39] Hao Zheng, Ahmed Louri, An energy-efficient network-on-chip design using reinforcement learning, in: Proceedings of the 56th Annual Design Automation Conference 2019, ACM, 2019, http://dx.doi.org/10.1145/3316781.3317768.
[40] Long-Ji Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, in: Reinforcement Learning, Springer US, 1992, pp. 69–97, http://dx.doi.org/10.1007/978-1-4615-3618-5_5.
[41] SPEC 2006 CPU. benchmark suite, http://www.tilera.com/. (Accessed 02 August 2021).
[42] Shubhangi D. Chawade, Mahendra A. Gaikwad, Rajendra M. Patrikar, Review of XY routing algorithm for Network-on-Chip architecture, Int. J. Comput. Commun. Technol. (2016) 271–275, http://dx.doi.org/10.47893/ijcct.2016.1384.
[43] R. S. Reshma Raj, C. Gayathri, Saidalavi Kalady, P.B. Jayaraj, Odd-even based adaptive two-way routing in mesh NoCs for hotspot mitigation, in: Proceedings of the 20th International Conference on Distributed Computing and Networking, ACM, 2019, http://dx.doi.org/10.1145/3288599.3288611.
[44] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, Vladimir Stojanovic, DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling, in: 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip, IEEE, 2012, http://dx.doi.org/10.1109/nocs.2012.31.

Reshma Raj R.S. is currently pursuing her Ph.D. in the Department of Computer Science and Engineering, National Institute of Technology Calicut, India. Earlier, she received her B. Tech in Computer Science and Engineering from Kerala University, India in the year 2014, and M. Tech in Computer Science and Engineering from Kerala Technological University, India, in the year 2017. Her research interests include multicore computer architecture, Network-on-Chip and machine learning.

Rohit R. received his Bachelor's degree in Computer Science and Engineering from National Institute of Technology Calicut in 2021. His research interests include Artificial Intelligence, Machine Learning, Network-on-Chip and Cloud Computing.

Mushrif Shaikh Shahreyar received Bachelor's in Computer Science and engineering from National institute of technology Calicut in 2021. His research interests include Network-on-Chip, Machine Learning, and Deep Reinforcement Learning.

Akash Raut received his Bachelor of Technology degree in Electrical and Electronics engineering from National institute of technology Calicut in 2021. His area of research include Deep learning, Machine learning, Deep Reinforcement Learning and Robotics.

Dr. **Pournami P.N.** is an Assistant Professor in the Department of Computer Science & Engineering, National Institute of Technology Calicut, Kerala, India. She has been an academician from last 10 years. She has been awarded PhD from National Institute of Technology Calicut, Kerala, India in the area of Digital Image Processing. Her research interests include Machine learning and Deep learning applications in Computer Vision, medical image analysis. She has published research papers in peer-reviewed journals with SCI and Scopus Indexed, and in international conference proceedings of Springer and IEEE.

Dr. **Saidalavi Kalady** is an Associate Professor in the Department of Computer Science and Engineering at National Institute of Technology, Calicut, Kerala, India. For the past 28 years he has been teaching in the field of Computer Science and Engineering. He was the Head of the Computer Science & Engineering department of National Institute of Technology, Calicut during 2016–2018. He obtained his PhD in the area of agent based systems from National Institute of Technology, Calicut. He completed Post Graduation from Indian Institute of Science, Bangalore, India. His research interests include Computer Architecture, Computational Intelligence and Operating Systems. He has published research papers in SCI indexed & Scopus indexed journals and conferences.

Dr. **Jayaraj P.B.** received his Ph.D. in Computer Science from National Institute of Technology Calicut, India. As part of his thesis, he had applied Machine Learning for simplifying the Drug Discovery pipeline. Presently, he is working as an assistant professor in the Department of Computer Science and engineering, NIT Calicut. His research interests include AI, High Performance Computing and Health informatics. He is author of a great deal of research studies published at national and international journals as well as conference proceedings.