J Supercomput (2014) 68:1214–1234
DOI 10.1007/s11227-014-1166-1

Adaptive load balancing in learning-based approaches for many-core embedded systems

F. Farahnakian · M. Ebrahimi · M. Daneshtalab · P. Liljeberg · J. Plosila

Published online: 28 March 2014

Abstract Adaptive routing algorithms improve network performance by distributing traffic over the whole network. However, they require congestion information to facilitate load balancing. To provide local and global congestion information, we propose a learning method based on dual reinforcement learning approach. This information can be dynamically updated according to the changing traffic condition in the network by propagating data and learning packets. We utilize a congestion detection method which updates the learning rate according to the congestion level. This method calculates the average number of free buffer slots in each switch at specific time intervals and compares it with maximum and minimum values. Based on the comparison result, the learning rate sets to a value between 0 and 1. If a switch gets congested, the learning rate is set to a high value, meaning that the global information is more important than local. In contrast, local is more emphasized than global information in non-congested switches. Results show that the proposed approach achieves a significant performance improvement over the traditional Q-routing, DRQ-routing, DBAR and Dynamic XY algorithms.

Keywords Congestion-aware routing algorithm · Adaptive routing algorithm · Q-learning and Q-routing approaches

1 Introduction

As the routing model impacts the performance and power consumption of the network in Many-core Embedded Systems, a considerable amount of research has been done

F. Farahnakian (✉) · M. Ebrahimi · M. Daneshtalab · P. Liljeberg · J. Plosila
University of Turku, Turku, Finland
e-mail: fahfar@utu.fi

to improve the routing efficiency of the network. Routing algorithm determines the output channels for forwarding a packet to the destination switch.

Generally, routing algorithms can be classified into deterministic and adaptive [1]. In deterministic routing algorithms, a transfer path is completely determined by the source and destination addresses. These algorithms are unable to alleviate congestion since the routing decision is independent of the network condition. For example, in the XY routing algorithm, packets first transfer along the *X* direction, then along the *Y* direction. Implementing deterministic routing is simple because it stays unchanged during network's operation. However, when the packet injection rate increases, deterministic algorithms cannot balance the load across the links. In addition, congestion is a major factor in limiting the performance of network and needs to be avoided [2]. In adaptive routing algorithms [3], the transfer path of each packet is determined based on the current network conditions. When network congestion happens, they choose paths with low latencies to avoid congested links and switches.

Reinforcement Learning (RL) [4] is a machine learning paradigm that has been widely applied in many different areas. In RL, an agent or decision maker learns a model of the environment at run-time and then utilizes these experiences to find an effective control policy for the given task. So, it provides a general framework for high-performance, self-optimizing controller design without prior knowledge. One of the most popular RL algorithms is Q-learning that has been applied in many different areas. In Q-learning [5], the learning agent percepts the environment and chooses an action at each state. The agent receives a reward after every action execution which indicates the performance of action. The final goal of the agent is to learn a policy for selecting the best action among all possible actions. The Q-learning algorithm was used to create an adaptive routing algorithm in the network communication named Q-routing [6]. Each switch makes its routing decisions based on the information on their neighboring switches in Q-routing. In other words, a switch stores a table of Q-values that estimate the quality of the alternative paths. These values are updated each time the switch sends a packet to one of its neighbors. Therefore, a few Q-values might be updated depending on the traffic patterns and load levels in the network. Dual Reinforcement Q-routing (DRQ-routing) [7] has been presented to address this problem. In DRQ-routing, the Q-values are updated by carrying the latency information to neighboring switches (backward exploration unique to DRQ-routing) and by receiving packets from neighboring switches (forward exploration similar to Q-routing). The speed of learning is expected to be higher in DRQ-routing than Q-routing because of the enhanced exploration in DRQ-routing. Therefore, the routing policy learned by DRQ-routing leads to a higher performance than Q-routing in terms of average packet delivery time at high and medium injection loads.

In this paper, we present a routing algorithm named Congestion-aware routing Algorithm using Dual Q-routing (CADuQ) with the following main contributions:

- A learning method based on DRQ-routing is utilized in CADuQ to alleviate congestion in the network by providing local and global congestion information. This congestion information is carried by packets throughout the network. Each switch maintains a table to store the estimated latencies from the source to any possible destination switches. The corresponding entry of the table is updated whenever

the switch receives a packet. This method employs both backward and forward explorations to update the estimated latency values in each switch. The estimated latencies are used to select less congested paths for packets.

- The waiting time in the packet queue of a switch is a commonly used metric for estimating latency in communication networks. Since the range of the waiting time varies from zero to infinity, the size of the tables should be very large to store the estimated waiting times. This is not a suitable approach due to its area overhead. Although an upper bound value can be considered, but this cannot be easily determined as it highly depends on the traffic condition in the network. In this paper, we considered the average number of free slots in input ports as a metric to reflect the congestion condition in each switch. In this way, not only the values are limited to a bounded range but also as shown in the result section, they lead to a better performance gain.
- In traditional methods, commonly, a static learning rate is used whatever the network load is. Another contribution of this paper is to adjust the learning rate in accordance with the changing traffic condition of the network. For this purpose, we present a congestion detection method which calculates the average number of free buffer slots for each switch at specific time intervals. According to the obtained value, when the network is congested, the learning rate is set to a large value to keep the values as updated as possible. In contrast, when the switch is not congested the learning rate is set to a minimum value amplifying the impact of local congestion values.

The reminder of this paper is organized as follows. In Sect. 2, the related work is discussed. Some background information on Q-routing and DRQ-routing techniques is given in Sect. 3. In Sect. 4, we discuss the congestion metrics that can be used to estimate the congestion status in the network. The congestion detection method is described in Sect. 5. The proposed adaptive routing algorithm is explained in Sect. 6. The results are reported in Sect. 7, while the summary and conclusion are given in the last section.

2 Related work

A significant amount of research has been directed towards developing adaptive routing protocols in network. Adaptive routing algorithms, depending on the degree of adaptiveness can be classified as partially adaptive or fully adaptive [8]. Partially adaptive routing algorithms use some of the shortest paths between the sender and the receiver, but not all packets are allowed to use every shortest path. For example, the turn model routing is based on prohibiting certain turns during the packet routing to prevent deadlock [9,10]. Fully adaptive routing algorithms allow routing packets on any shortest paths. To make the routing algorithm fully adaptive, there is a need of virtual channels in a wormhole switching network [11]. Adaptive routing algorithms can be distinguished whether they are minimal or non-minimal. Minimal adaptive routing algorithms only route packets along shortest paths to their destinations. Non-minimal routing algorithms do not necessarily use shortest paths. Adaptive routing policies can be categorized into congestion-oblivious and congestion-aware

schemes. In congestion-oblivious algorithms, routing decisions are independent of the congestion condition of the network. Random [13] and Zigzag [12] are two examples of congestion-oblivious methods. DyXY [14] and DyAD [15] are two approaches of the congestion-aware routing algorithms. These routing algorithms consider the congestion status of the network in the routing decision. The congestion level of a switch can be measured based on the number of available virtual channels [16] or queue length [17] at input port. The presented approach in [18] uses the amount of free buffer space available in each virtual channel of adjacent switches as congestion metric. Congestion-aware routing policies can be further classified based on whether they rely on purely local congestion information or take into account the congestion status at other points in the network. In DyXY, a packet is sent either to the X or Y direction depending on the congestion condition. It uses local information which is the current queue length of the corresponding input port in the neighboring switches to decide on the next hop. Most common implementations of routing algorithms, e.g., NoP [19], RCA [20], DBAR [21], and CATRA [22], have focused on collecting local or global congestion information to get an estimation of the congested areas in the network.

The Q-routing allows a network to be constantly adapted to the changing traffic condition. In this method, each switch makes its routing decision based on Q-values which estimate the quality of alternative routes. However, Q-routing suffers from the unreliability of the estimated Q-values. The reason is that depending on traffic patterns and load levels, only few Q-values are updated regularly while most of the Q-values in the network remain un-updated. Some proposals have been presented to address this issue, such as PQ-routing [23], CQ-routing [24] and DRQ-routing [25]. PQ-routing keeps the best Q-values and reuses them by predicting the traffic trend. This algorithm is able to explore paths that have been inactive for a long time, and thereby is able to restore its previous policy. In CQ-routing, each Q-value is attached with a confidence value (C-value) which determines how closely the corresponding Q-value represents the current state of the network. A similar idea is presented in DRQ-routing where the speed of restoration is expected to be high. In DRQ-routing, learning is performed by the latency information appended to data packet to neighboring switches (backward exploration) and also by receiving latency information from the neighboring switch where a data packet is sent to (forward exploration). This approach leads to increase the speed of learning by providing a bi-directional exploration.

Reinforcement learning approaches have rarely been investigated in on-chip networks. A fault-tolerant deflection routing algorithm is presented in [26] which is inspired by Q-learning techniques for tolerating faults. In another work, DyNoC [27] is proposed to handle communication among modules which are dynamically placed on a reconfigurable on-chip network. Q-routing is also used to provide different levels of Quality-of-Service (QoS) such as Best Effort (BE) and Guaranteed Throughput (GT) [28]. In this approach, a novel scheme is presented which contrasts the performance of Q-routing with the XY routing strategy in the context of QoS. C-routing [29] is a Q-learning-based method which takes advantage of clustering approach to reduce the routing table size compared with conventional Q-routing algorithms. A Q-learning-based Congestion-aware Algorithm (QCA) [30] is presented to alleviate

congestion condition. QCA estimates and predicts the congestion condition of the network as close to the actual values as possible. This estimation is used in the routing decision to choose a less congested path. Moreover, Dual Q-routing Adaptive learning Rate [31] has enhanced Q-routing performance for Networks-on-Chip when the network becomes congested. For this to happen, a congestion detection technique is introduced in this method which updates information dynamically according to the changing traffic condition. HARAQ [32] takes the best usage of allowable turn in the network and finds all alternative paths to route packets. Then, it uses a learning approach to find an optimal path between all options of minimal or non-minimal routes. However, Q-learning approach suffers from high area overhead in NoCs due to the need for a large routing table in each switch. To reduce the area overhead, we have presented a clustering approach in previous works [33,34] that decreases the number and size of routing tables.

3 Background

This section reviews the basic concepts of Q-learning which is one of the most widely used reinforcement learning methods. Then, we describe Q-routing algorithm by applying Q-learning to a communication network. Finally, we explain the DRQ-routing approach taking advantage of the dual reinforcement learning aspect.

3.1 Q-learning

Q-learning [5] is a highly adaptive algorithm which enables an agent or decision maker to interact with the environment without prior knowledge about the system status to achieve a given goal. At each step of an episode, the agent first observes the current system state (s) and chooses an action (a). By performing the action, the system moves to the next state (s') and the agent also receives a reward (a real number) or punishment (a negative real number). The reward value updates the Q-value, $Q(s, a)$, which represents the expected long-term reward of taking action a in state s. Q-values are stored into a Q-table that maintains four fields named Current State, Next State, Action, and Q-value. *Current State* refers to a state of environment can be observed by the agent, while *Next State* is determined by the *Current State* and the actions selected by the agents. The *Action* indicates the action that the agent performs. *Q-value* holds the reward value that the agent receives in the current state when it executes an action.

3.2 Q-routing

Q-routing [6] is an adaptive routing method based on the Q-learning model in a communication network. Each switch includes a Q-table that estimates the quality of alternative routes. Q-table is updated each time a switch sends a packet to one of its neighbors. Assume that a packet destined for the switch d is forwarded from switch

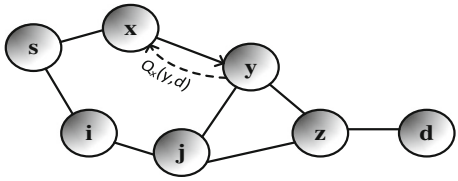


Fig. 1 An example of Q-routing method

x to the neighboring switch y, (Fig. 1). The maximum time it will take for a packet to reach its destination from the switch x is bounded by the sum of three quantities:

- The waiting time (q_y) in the packet queue of the switch y.
- The transmission delay (δ) over the link from switch x to y.
- The time $Q_y(z, d)$ it would take for the switch y to send this packet via the switch z to the destination switch (Fig. 1). Switch z has the minimum latency (Q-value) between the neighbors of switch y. $Q_y(z, d)$ is obtained from the following equation:

$$Q_y(z, d) = \min_{n \in N(y)} Q_y(n, d)$$

  where $N(y)$ is a set neighbors of switch y.

The switch y sends an estimated latency value to the switch x. This value contains the best latency estimate from the switch y to the destination $d(Q_y(z, d))$ and the waiting time at the input buffer of switch $y(q_y)$ back to the switch x. Upon receiving the estimated value, the switch x computes the new estimate for $Q_x(y, d)$ as follows:

$$Q_x(y, d)_{\text{est}} = Q_y(z, d) + q_y + \delta$$

$Q_x(y, d)_{\text{est}}$ is the switch x's best estimated delay that it would take for a packet to reach its destination switch d from the switch x when sent via its neighboring switch y. This value updates the $Q_x(y, d)$ by the following in Q-table and the learning is performed through updating Q-values.

$$Q_x(y, d)_{\text{new}} = Q_x(y, d)_{\text{old}} + \gamma(Q_x(y, d)_{\text{est}} - Q_x(y, d)_{\text{old}}) \quad (1)$$

Learning rate (γ) determines in which rate the new information overwrites the old one. Learning rate can take a value between zero and one; the value of zero means that no learning takes place by the algorithm, while the value of one indicates that only the most recent information is used.

The Q-routing algorithm has two steps as follows:

- Step 1: The switch x sends a packet, destined for the switch d, to one of its neighboring switches, y.

1. *Select a packet from the queue.*
2. *Find the minimum Q-value from the switch x to the destination switch d, through one of the neighboring switches, assumed y.*

$$Q_x(y,d) = \min_{m \in N(x)} Q_x(m,d)$$

where N(x) is a set of the x's neighboring switches.

3. *Forward the packet to the neighboring switch y.*
4. *As the switch y receives a packet from switch x, find the minimum Q-value from the switch y to the destination switch d through one of the neighboring switches, assumed z.*

$$Q_y(z,d) = \min_{n \in N(y)} Q_y(n,d)$$

where N(y) is a set of the y's neighboring switches.

5. *Send the forward latency estimation back to switch x which includes $Q_y(z,d)$ and q_y.*
6. *Get ready for receiving the next packet (goto 1).*

– Step 2: When the switch x receives the forward estimation from its neighboring switch y.

1. *Switch x receives an estimated value from neighbor y containing $Q_y(z,d)$ and q_y.*
2. *Update the Q-value ($Q_x(y, d)$) as given in Equation (1).*

During the first times of the learning process, Q-routing learns a representation of the network state in terms of Q-values. Then, it uses these values to make routing decisions and find the least congested path among available paths from a source switch to a destination switch.

3.3 DRQ-routing

The Q-routing method only updates the Q-values when a switch sends a packet to neighboring switches. Therefore, this algorithm only uses forward exploration to determine the latency of the remaining path from the current switch to the destination switch. The DRQ-routing technique utilizes Q-routing for both backward and forward exploration for updating Q-values more frequently. Backward exploration indicates the latency of the traversed path from the current to the source switch. Therefore, the corresponding entries of the Q-table are updated with both forward and backward exploration rather than only forward exploration in Q-routing. As an example of the backward exploration, consider a case where the switch x sends a packet to its destination switch d via its neighboring switch y. When the data packet traverses from the source to the destination, it carries some latency information between each two neighboring switches. As the switch y receives this packet, it uses this information for updating its own estimate of sending a packet to source s via switch x.

Suppose the packet is currently at switch x as in Fig. 2, it contains the minimum latency value from the switch s to the switch x passing though the switch h. This value can be defined as:

$$Q_x(h,s) = \min_{n \in N(x)} Q_x(n,s)$$

When the packet arrives at switch y, it can update the estimation of sending a packet to the switch s via neighbor x. The new value includes $Q_x(h,s)$ and the waiting time

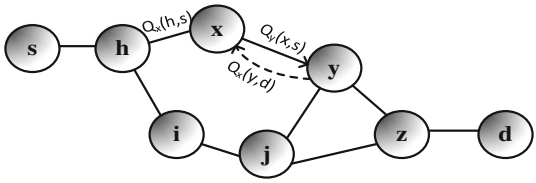


Fig. 2 An example of DRQ-routing method

at the input buffer of switch $x(q_x)$:

$$Q_y(x,s)_{\text{new}} = Q_y(x,s)_{\text{old}} + \gamma((Q_x(h,s) + q_x + \delta) - Q_y(x,s)_{\text{old}}) \quad (2)$$

In this way, each packet carries the routing information from a source to a destination. These values are used to update the Q-values of intermediate switches. In other words, the receiving switch uses the latency information of the traversed path to update the Q-values.

4 Congestion detection metrics

The primary objective of the proposed algorithm is to improve the network performance by routing the packets through the less congested paths. In Q-routing, Q-values reflect the real congestion level in each switch. So the congestion can be alleviated in the network by sending packets through paths with minimum Q-values. For calculating the Q-values, we consider two metrics:

Waiting time (wTime): when a header flit enters a switch through one of the input port, it is stored in an input buffer. The flit is kept in a switch until it proceeds to the routing unit. If the switch is congested, the flit potentially waits on the input channel for extended period of time. Therefore, the waiting time in the input channel can determine the congestion status in a switch. *wTime* metric indicates the time from when a header flit enters an upstream switch until it is processed by the routing unit.

Average free buffers (AvgBf): indicates the average number of free input buffer slots at downstream switches. The count of free buffers was first proposed as an indicator of congestion by Kim et al. [18]. Higher buffer capacity or a larger number of free buffer slots in each virtual channel will reduce network contention, thereby reducing latency.

Measuring Q-values based on these metrics can reflect the congestion level of each switch. To validate the efficiency of congestion metrics, we set up experimental environment using a wormhole-based simulator based on the OMNET framework [35]. For all switches, the data width is set to 32 bits and the maximum bandwidth at each link is 1 flit per cycle. We used two virtual channels that each input buffer (FIFO) size of 8 flits. In each simulation time step (nanosecond), these packets are stored in the queue of the source switches. The amount of injected packets depends on the network offered load. The network offered load is calculated as follow:

$$\text{Network offered load} = \text{flit-size}/\text{flit arrival delay}$$

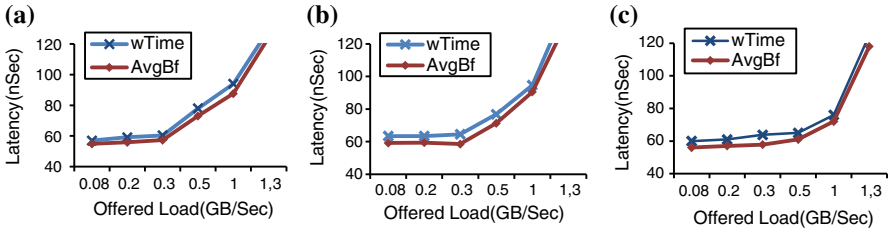


Fig. 3 Performance evaluation under **a** uniform, **b** transpose and **c** hotspot traffic in the 8×8 mesh network

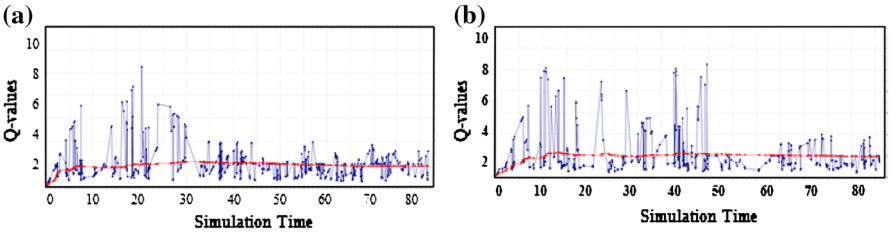


Fig. 4 The variation of Q-values based on the **a** AvgBf **b** wTime metric of a hotspot switch at (4, 4) in the 8×8 2D-mesh

where *flit_size represents* the size of the flit and *flit-arrival-delay* represents the delay time between previous flit generation and next flit generation.

The simulations were conducted on the 2D-mesh which is the most typical topology. The mesh size is selected to be 8×8 in our experimental results. The average latency as a function of the offered load is plotted for uniform random, transpose and hotspot traffic patterns. Experimental results (Fig. 3) show that Q-values based on the average number of free buffers (Avg*Bf*) are more efficient than *wTime* under various traffic patterns. It can be obtained that Avg*Bf* represent the real latency value.

To demonstrate how fast the Avg*Bf* can detect the network congestion, Fig. 4 depicts the learning process under the hotspot traffic with a single hotspot switch at (4, 4) in the 8×8 2D-mesh. Figure 4a and b show the Q-values when using the Avg*Bf and wTime* metrics, respectively. The variation of Q-values is plotted on the vertical axis and simulation time on the horizontal axis. The variations of Q-values in Fig. 4a are much less than that of Q-values in Fig. 4b. Moreover, Fig. 4a shows that the Q-values are converged faster and have a shorter learning process. Because the variation of Q-values is almost stable and the network is fully trained after 30.000 nanosecond (ns) in Fig. 4a. As shown in Fig. 4b, the variation of Q-values reaches to a stable phase after 45.000 ns in 8×8 mesh. As can be obtained from these two figures, the Avg*Bf* can reflect the congestion status of the network efficiently. The red line in these plots shows the standard deviation of the Q-values.

5 Congestion detection method

To distribute the traffic, we need to distinguish the current congestion level of the network. Results in Sect. 4 show that using *AvgBf* for Q-values is more efficient than *wTime* to represent the current congestion status of the network. Furthermore, it is nec-

essary to update the Q-values dynamically based on the network congestion condition. For this purpose, we present a congestion detection method which calculates the average number of free buffer slots of each switch at a predetermined time interval. After that, the average number of free buffer slots is compared with the maximum *(MaxVal)* and minimum *(MinVal)* values at each time interval. A massive experimental result was done to estimate the *MaxVal* and *MinVal* values. Based on our analysis, in general, the best results were obtained when *MinVal* and *MaxVal* set to 25 and 65 %, respectively, of total buffer slots. Thereby, a switch is counted as congested one if $\leq$25 % of total queue size is free (*MinVal*). On the other hand, a switch is counted as a less congested one if more than or equal to 65 % of total buffer size in a switch is free (*MaxVal*). The Eqs. (3) and (4) are used to set the *MinVal* and *MaxVal* values, respectively.

$$MinVal = 25\ \% \times (TotalBufferSlots) \tag{3}$$

$$MaxVal = 65\ \% \times (TotalBufferSlots) \tag{4}$$

If *AvgBf* in a switch is larger than *MaxVal*, it indicates that the switch is not congested and it is unnecessary to update the Q-values regularly. Thus, the learning rate is set to a minimum value amplifying the impact of local congestion statuses (*LearnRate* = 0.1). Moreover, if *AvgBf* in a switch is smaller than *MinVal*, the learning rate is set to a large value to keep the Q-values as updated as possible (*LearnRate* = 0.9). Finally, if *AvgBf* is between *MinVal* and *MaxVal, LearnRate* is set to 0.5. In this way, global congestion information gets more emphasis than local values. The functionality of congestion detection method is given as follow:

```
---congestion detection method
1.  MinVal=25% (TotalBufferSlots);
2.  MaxVal=65% (TotalBufferSlots);
3.  AvgFreeBufferSlots, count_i =0;
4.    for  t = current simulation time  to 200ns OR 100 clk then
5.          if  switch_i  receives a flit
6.               count_i=count_i+1;
7.               FreeBufferSlots_i+= FreeBufferSlots_i[t];
8.         end if;
9.    end for;
10. AvgFreeBufferSlots_i= FreeBufferSlots_i /count_i ;
-- Determine the learning rate
11. if  AvgFreeBufferSlots_i< = MinVal
12.      LearnRate= 0.9;
13. else if  MinVal < AvgFreeBufferSlots_i< MaxVal
14.      LearnRate = 0.5;
15. else if  AvgFreeBufferSlots_i>= MaxVal
16.      LearnRate= 0.1;
```

Initially, *AvgBf* is set to zero. Then it is measured in 200 ns (100 cycles) time interval (line 4–10) and then it compares with the *MinVal* and *MaxVal* values. Finally, the learning rate is set to a new value which is used during the current time interval (line 10–17).

Throughout our experiments, we have considered learning rate values. These values are between 0 and 1 and control how much to change the Q-values. We could miss it in Eqs. (1) and (2) by setting γ to 1. The switch takes longer to learn when this rate set to a large value and Q-values update frequently. We, therefore, use three values learning rate depending upon the congestion level in a switch.

In general, if the network gets congested, the Q-values are frequently updated and global congestion values from distant switches become reliable. In contrast, a switch may receive few packets in a time interval, so that the values from distant switches are not accurate and the local values should be more emphasized than the global ones.

6 Congestion-aware routing algorithm using DRQ-routing

In this section, we describe the routing algorithm based on the adaptive learning rate. We explain the format of Q-tables according to the 2D-mesh network features along with the format of packets that propagate in the network. Two types of packets can travel within the network, data packets and learning packets. Data packets carry both data and congestion information (used in backward exploration) while learning packets carry only the congestion information (used in forward exploration). Since the learning packets can increase the latency of data packets, we consider the separate Virtual Channels (VC) for transmitting the learning packets.

6.1 Routing algorithm

Routing algorithm selects an output channel for forwarding a packet to a destination switch. CADuQ method can efficiently estimate the latency of a packet to reach different destinations through each of the possible output channels. CADuQ diagnoses the congested path in the network and routes packets through the less congested ones. CADuQ has three main characteristics: (1) local and global congestion information is provided based on dual reinforcement learning by propagating data and learning packets. (2) Local congestion information used in forward exploration is measured according to the free input buffer slots of the input buffer. This information is carried by learning packets. However, local congestion information used in backward exploration is calculated based on the free buffer slots of the input buffer located in the output direction. The reason is that, this information is used when a packet traverses in the reverse direction (i.e., from the current node to the source node). This information is carried by data packets. (3) Since the reliability of latency values depend on the speed of learning, we utilized a congestion detection method for updating the learning rate that was described in Sect. 5. In addition, forward and backward explorations in the CADuQ can reflect the current congestion condition in the network. The CADuQ algorithm can be summarized in three steps as follows:

– Step 1: When a data packet is delivered from switch x to switch y.

1. *Switch x determines the neighboring switch (y) with minimum latency value to the destination switch d:*
$$y = \min_{m \in N(x)} Q_x(m,d)$$
2. *Find the minimum estimated latency from the switch x back to the source switch s through the neighboring switch h ($Q_x(h,s)$):*
$$Q_x(h,s) = \min_{n \in N(x)} Q_x(n,s)$$
Also, find the average number of free output buffer slots in switch x (q_x).
--- Include the backward estimation ($Q_x(h,s)$ and q_x) into the header of the data packet.
3. *BackwardLatency* $\leftarrow Q_x(h,s) + q_x$
4. *Forward the packet to the neighbor y.*

– Step 2: When switch y receives a data packet from switch x.

1. *Extract the backward estimation including $Q_x(h,s)$ and q_x from the received packet.*
2. *Set the learning rate according to the "Congestion Detection" method.*
3. *Update the Q-value ($Q_y(x, s)$) using Equation (2).*
4. *Find the minimum latency estimation from the switch y to the destination switch d through the neighboring switch z:*

$$Q_y(z,d) = \min_{n \in N(y)} Q_y(n,d)$$

Also, calculate the average number of free input buffer slots in switch y (q_y).

5. *Send the learning packet back to the switch x including forward estimation ($Q_y(z,d)$ and q_y).*
6. *ForwardLatency* $\leftarrow Q_y(z,d) + q_y$

– Step 3: When a learning packet is transferred from switch y to switch x.

1. *Extract the forward estimation containing $Q_y(z,d)$ and q_y.*
2. *Set the learning rate according to the "Congestion Detection" method.*
3. *Update the Q-value ($Q_x(y, d)$) using Equation (1).*

6.2 Routing tables

In Q-routing, each switch has a Q-table similar to Fig. 5a that maintains the routing latency from itself to the possible destination switches. The Q-table contains n entries, where n is the number of switches in the network. In this table, each row corresponds to a destination; each column relates to an output port; the contents of the table are the estimated latencies. Therefore, routing unit selects an output port from the Q-table which has minimum latency value for a corresponding destination. Notice that, in conventional Q-routing models, the waiting time at input buffers is a measure of latency, however, we use the number of free buffer slots at input buffers as a metric of latency.

Since CADuQ is a minimal routing approach, packets can be delivered into at most two directions at a switch. Our basic structure of the adjusted Q-table is shown in Fig. 5b in which the number of columns is reduced to two. Therefore, the table size is decreased by 50 % compared with conventional Q-tables.

6.3 Packets format

The header flit of data packet contains routing information such as destination and source addresses. In CADuQ, the data packet format is modified by adding five bits into the header flit (Fig. 6). These additional fields can be described as follows:

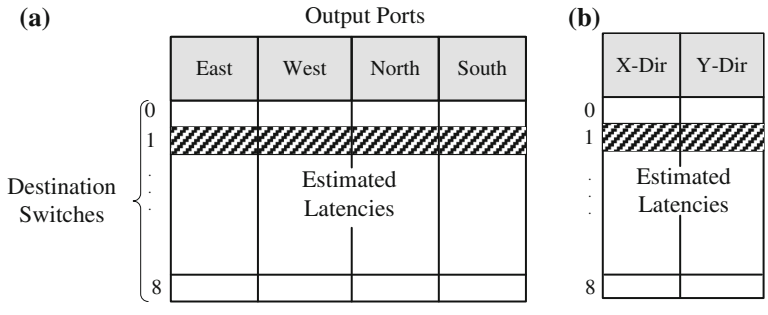


Fig. 5 **a** Conventional Q-table, **b** adjusted Q-table of switch 1 in 3 × 3 mesh network

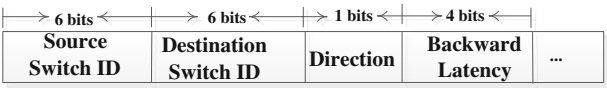
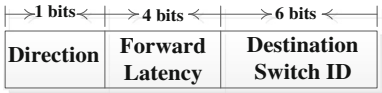


Fig. 6 Header of the data packet format

Fig. 7 Learning packet format



- *Direction*: determines the direction in which the data packet is delivered from the current switch. The value of 0 and 1 are corresponding to the *X*-direction and *Y*-direction, respectively. Since the packet is sent either to the *X* or *Y* dimension, one bit is enough to encode the direction of neighboring switch.
- *Backward Latency*: determines the expected total latency to forward a packet from the current switch to the source switch. In Eq. (2), the terms $Q_x(h, s)$ and q_x indicate the global and local information of the backward latency. Total latency values are determined with four bits.

The format of the learning packet is illustrated in Fig. 7, which consists of three fields as follows:

- *Direction*: It is a one bit value determining the direction of receiver switch of the data packet or direction of sender switch of the learning packet.
- *Forward Latency*: It is obtained by summing up the forward local and global latency values. Local latency is measured by the average free buffer slots in downstream switch. In Eq. (1), the values of $Q_y(z, d)$ and q_y indicate the forward local latency. The term $Q_y(z, d)$ in Eq. (1) indicates the global and local information in forward latency. This latency value determines the expected latency of a packet from the next switch to the destination. We have assigned four bits to represent the forward latency.
- *Destination switch ID*: It determines the destination switch of a data packet. Six bits are considered for this purpose which is sufficient for 8×8 mesh network. For a bigger size network, more bits should be allocated to the destination field.

Since learning packets only traverse between neighboring routers, they do not require any routing process. In addition, the size of learning packets is smaller than data packets which results in wasting resources if sharing resources with data packets.

Now, we present an example for the CADuQ routing process in a 3×3 mesh network. The number of VCs is two and each VC has a buffer where buffer size is set to 8 flits. The *MinVal* and *MaxVal* value can be computed using Eqs. (3) and (4):

$$MinVal = 25\ \% \times 16 = 4$$
$$MaxVal = 65\ \% \times 16 = 10.4$$

In Fig. 8, suppose a packet is generated at the source switch 0 for the destination switch 8. Two output channels (East and North) can be selected at switch 0 for forwarding

the data packet to the destination. As illustrated in Fig. 8a, the east direction has the lowest latency to reach the destination switch, so it is chosen as the next switch. Before forwarding the packet to switch 1, the sum of backward local and global latencies is added to the header flit of the data packet. The backward local latency (7) is obtained from the Q-table, indicating the average number of free buffer slots at the output port of switch 0. The global latency is equal to zero because the switch 0 is the source switch. When the intermediate switch 1 receives the data packet from the switch 0, backward exploration information is extracted from the data packet (Fig. 8b). Based on this information, the first row and *X*-Dir column of the routing table in switch 1 are updated. Suppose the average free buffer slot of switch 1 is greater than *MaxVal*. So, the congestion condition of switch 1 is low and the learning rate is set to 0.1. This learning rate is used to update the Q-values. Therefore, according to Eq. (2), the new Q-value is:

$$\begin{aligned} Q_1(0,0)_{\text{new}} &= Q_1(0,0)_{\text{old}} + 0.1((0+q_0+\delta) - Q_1(0,0)_{\text{old}}) \\ &= 2 + 0.1((0+7+0)-2) = 2.5 \end{aligned}$$

We suppose that the link delay, δ, is a constant value. In this work, we set it to 0.

Using Q-table information at switch 1, among north and east directions, the north direction has the lowest estimated latency. Hence, switch 4 is selected for forwarding the packet to switch 8. At this time, the learning packet is generated to return the forward local and global latencies back to switch 0. Local information is the average number of free buffer slots at input port of switch 1. This value is assumed to be 6 in our example. The minimum estimated latency of routing the packet from switch 1 to switch 8 via the north port is considered as the global latency. This value is extracted from the Q-table in switch 1 (i.e., global = 3). The sum of the local and global values is the new latency estimation from switch 1 to the switch 0 which will be stored in the learning packet. Finally, the table in the switch 0 is updated whenever the learning packet is received. Assume the average number of free buffer slots in the switch 0 is 7. Since this value is between *MaxVal* and *MinVal* in the last time interval, the learning rate γ is set to 0.5.

$$\begin{aligned} Q_0(1,8)_{\text{new}} &= Q_0(1,8)_{\text{old}} + 0.5((Q_1(4,8)+q_1) - Q_0(1,8)_{\text{old}}) \\ &= 2 + 0.5((3+6)-2) = 5.5 \end{aligned}$$

As shown in Fig. 8a, the corresponding entry of the Q-table at switch 0 is updated taking the new estimated value and an existing estimation using Eq. (1). As can be seen from Fig. 8b, switch 4 is selected for forwarding the packet to the destination switch and a similar process is done until the packet reaches switch 8.

7 Results

To evaluate the performance of our proposed routing algorithm, we compare it with conventional on-chip network (DBAR and Dynamic XY) algorithms and Q-learning schemes (Q-routing and DRQ-routing). The performance of the routing scheme is

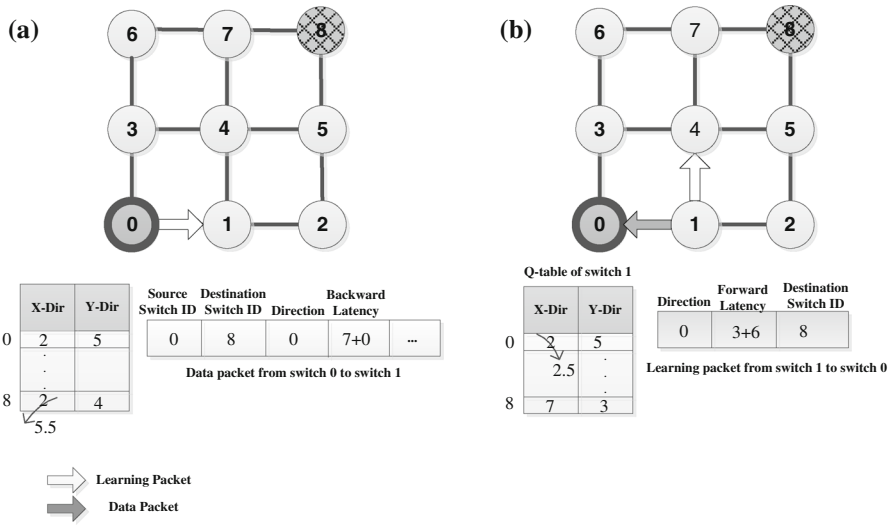


Fig. 8 An example of CADuQ for a 3 × 3 mesh

evaluated through latency curves. The simulations were conducted on the 8 × 8 and 14 × 14 2D-mesh under various traffic patterns. It is assumed that latency is the duration from the time when the first flit is created at the source core, to the time when the last flit is delivered to the destination core. To propagate data packets, two VCs are used along the *X* and *Y* dimensions, while for learning packets, one virtual channel is utilized. The buffer size per VC is set to eight flits. The simulator is warmed up with 3,000 packets and then the average performance is measured over 10,000 data packets. Three synthetic traffic profiles including uniform random, transpose and hotspot, along with SPLASH-2 [36] application traces are used.

7.1 Uniform random traffic profile

In uniform random traffic, each core sends a packet to another core with a random probability. The destination of different packets in each switch is determined randomly using a uniform distribution. In Fig. 9, the average latency as a function of the offered load is plotted. As observed from the results, the proposed routing scheme achieves better performance compared with the other schemes. As load increases, DBAR is unable to tolerate the high load condition, while Q-routing schemes learn an efficient routing policy. CADuQ can alleviate the congested areas and perform considerably better than other schemes.

7.2 Transpose traffic profile

Transpose traffic means that a switch with the coordinates (i, j) sends packets to switch with the coordinates $(n - j, n - i)$ in an $n \times n$ mesh. Figure 10 shows CADuQ behaves

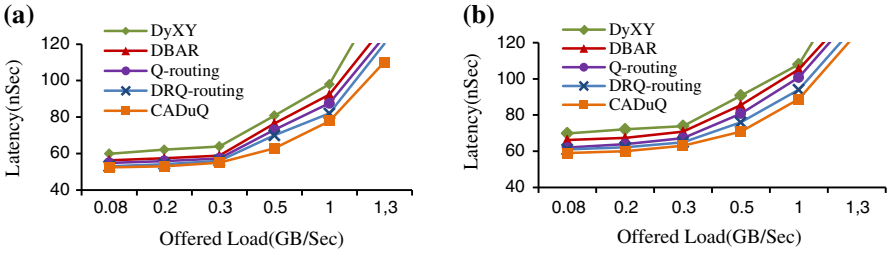


Fig. 9 Performance under different traffic models in **a** 8 × 8 2D-mesh and **b** 14 × 14 2D-mesh under the uniform traffic model

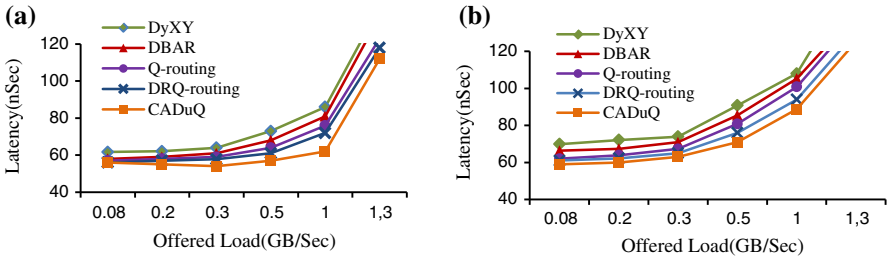


Fig. 10 Performance under different traffic models in **a** 8 × 8 2D-mesh and **b** 14 × 14 2D-mesh under the transpose traffic model

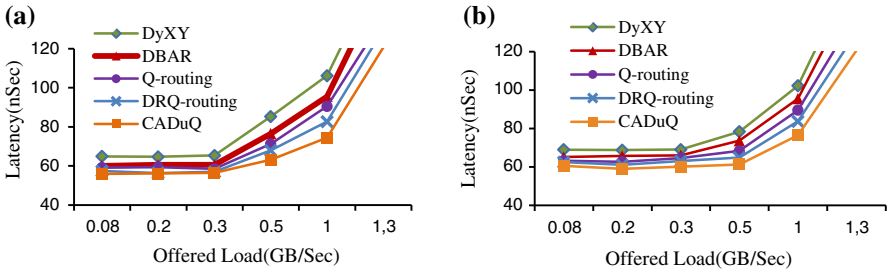


Fig. 11 Performance under different traffic models in **a** 8 × 8 2D-mesh and **b** 14 × 14 2D-mesh under the hotspot traffic model

as efficiently as other routing algorithms. CADuQ leads to the lowest latency due to the fact that it can increase the exploration (twofold) by dual reinforcement learning. In turn, increased exploration leads to increased speed of learning. Moreover, the proposed method improves the quality of learning by adapting learning rate based on the congestion condition.

7.3 Hotspot traffic profile

Under the hotspot traffic pattern, one or more switches are chosen as hotspots receiving an extra portion of the traffic in addition to the regular uniform traffic. This traffic represents a more realistic traffic pattern. In simulations, given a hotspot percentage

Table 1 Performance gain of the CADuQ method over others methods for three traffic patterns in a 8 × 8 2D-mesh

Traffic pattern	Q-routing (%)	DRQ-routing (%)	DBAR (%)	DyXY (%)
Uniform	17.7	12.9	24.1	30.6
Transpose	12.2	7	19.2	28
Hotspot	14.2	8	23.8	35

Table 2 Performance gain of the CADuQ method over others methods for three traffic patterns in a 14 × 14 2D-mesh

Traffic pattern	Q-routing (%)	DRQ-routing (%)	DBAR (%)	DyXY (%)
Uniform	15.4	8.6	22.8	30
Transpose	9.6	7.2	12.7	20.1
Hotspot	11.6	9.4	18.3	26.7

of H, a newly generated packet is directed to each hotspot switch with an additional H percent probability. We simulate the hotspot traffic with a single hotspot switch at (4, 4) and (7, 7) in the 8 × 8 and 14 × 14 2D-meshes, respectively. The average latency of each network with H = 10 % are illustrated in Fig. 11. As observed from the figure, the Q-learning schemes behave as efficiently as DBAR and DyXY especially in medium and high loads. Figure 11 illustrates the performance gain of our proposed method over Q-routing, DRQ-routing and conventional on-chip network (DBAR and Dynamic XY) algorithms near the saturation point (0.5) for a 14 × 14 2D-mesh. Using minimal routes along with the intelligent selection policy reduces the average network latency of CQ-routing for both mesh sizes.

Table 1 illustrates the performance gain of the proposed approach over Q-routing, DRQ-routing, DBAR and Dynamic XY algorithms near the saturation point (0.5) for a 8 × 8 2D-mesh. Experimental results under different traffic patterns demonstrate that the on-chip network utilizing the CADuQ routing method clearly outperforms the conventional approach considerably.

In addition, the impact of using dual reinforcement learning policy and congestion detection method near the saturation point (0.5) over the DyXY routing, DBAR and the other Q-routing techniques is summarized in Table 2 for a 14 × 14 2D-mesh. The results reveal that the proposed approach can distribute the traffic efficiently.

7.4 Application traffic profile

Application traces are obtained from the GEMS simulator [37] using some application benchmark suites selected from SPLASH-2. We use a 64-switch network configuration: 20 processors and 44 L2-cache memory modules. For the CPU, we assume a core similar to Sun Niagara and use SPARC ISA. Each L2 cache core is 512 kB, and thus, the total shared L2 cache is 22 MB. The memory hierarchy implemented is governed by a two-level directory cache coherence protocol. Each processor has a private write-

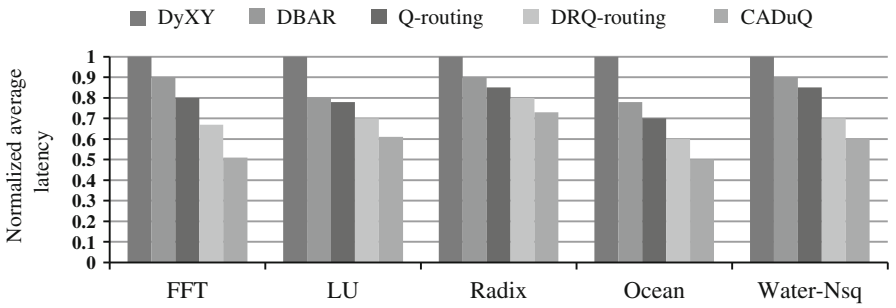


Fig. 12 Performance across SPLASH-2 benchmarks

Table 3 Hardware cost

Routing method	Network area (mm^2)
DyXY	0.1503
Q-routing	0.1683
DRQ-routing	0.1689
CADuQ	0.1705

back L1 cache (split L1 I and D cache, 64 kB, 2-way, 3-cycle access). The L2 cache is shared among all processors and split into banks (44 banks, 512 kB each for a total of 22 MB, 6-cycle bank access), connected via on-chip routers. The L1/L2 block size is 64 B. Our coherence model includes a MESI-based protocol with distributed directories, with each L2 bank maintaining its own local directory. The simulated memory hierarchy mimics SNUCA while the off-chip memory is a 4 GB DRAM with a 220-cycle access time. Figure 12 shows the average packet latency across five benchmark traces, normalized to DyXY. CADuQ provides lower latency than other schemes and it shows the greatest performance gain on Ocean with 50 % reduction in latency.

7.5 Area overhead

To estimate the hardware cost of our proposed method along with other routing schemes, the on-chip router of each scheme is implemented with VHDL and synthesized with Synopsys Design Compiler using the 65 nm standard CMOS technology with a timing constraint of 1 GHz for the system clock and supply voltage of 1 V.

The synthesized netlist is verified through post-synthesis simulations. The layout areas of the four schemes are listed in Table 3 and the area overhead of CADuQ is verified with the other learning methods. However, the result shows that the proposed approach increases the area overhead, but it can be decreased by applying clustering approach in [33,34].

8 Conclusion

In this paper, we propose a congestion-aware routing algorithm based on dual reinforcement learning for multi-processor platform. Commonly, in learning approaches,

each switch maintains a routing table which is updated at run-time by local and global latency information (Q-values). The latency values are obtained when transferring data and learning packets between adjacent switches. Depending on the network load and traffic patterns, some of the Q-values may not be updated for a long time. On the other hand, the routing decision based on unreliable Q-values cannot be accurate and does not necessarily reflect the current congestion state of the network. For addressing this issue, we proposed a technique to adjust the learning rate with the network condition. A congestion detection technique is utilized to compute the current congestion level of a switch. If a switch is congested, the learning rate is set to a large value to keep the global latency values as updated as possible. Otherwise, when the switch is not congested, a small value is assigned to the learning rate, meaning that the local information prioritize over global information. The experiments show that the CADuQ routing method is able to route packets more efficiently than traditional methods. For instance, the performance is improved up to 35 % compared with the DyXY method under hotspot traffic in 8×8 mesh network.

References

1. Ni LM, McKinley PK (1993) A survey of wormhole routing techniques in direct networks. Computer 26(2):62–76
2. Ebrahimi M, Daneshtalab M, Liljeberg P, Plosila J, Tenhunen H (2011) Agent-based on-chip network using efficient selection method. In: Proceedings of 19th IFIP/IEEE International Conference on very large scale integration (VLSI-SoC), pp 284–289.
3. Dehyadegari M et al. (2011) An adaptive fuzzy logic-based routing algorithm for networks-on-chip. In: Proceedings of 13th IEEE/NASA-ESA International Conference on adaptive hardware and systems (AHS), pp 208–214.
4. Sutton RS, Barto AG (2000) Reinforcement learning. MIT Press, Cambridge, An introduction
5. Watkins CJCH, Dayan P (1992) Q-Learning. In: Proceedings on machine learning, pp 279–292.
6. Boyan JA, Littman ML (1994) Packet routing in dynamically changing networks: a reinforcement learning approach. Adv Neural Inf Process Syst 6:671–678
7. Kumar S, Miikkulainen R (1997) Dual reinforcement Q-routing: an on-line adaptive routing algorithm. In: Proceedings of the artificial neural networks in engineering Conference, pp 231–238.
8. Schonwald T, Zimmermann J, Bringmann O (2007) Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures. In Euromicro Conference on digital system design architectures, methods and tools (DSD), Lübeck, pp 527–534.
9. Chiu G-M (2000) The odd-even turn model for adaptive routing. IEEE Trans Parallel Distrib Syst 11(7):729–738
10. Ebrahimi M et al. (2012) MAFA: adaptive fault-tolerant routing algorithm for networks-on-chip. In: Proceedings of 15th IEEE Euromicro Conference on Digital System Design (DSD), pp 201–206.
11. Boura YM, Das CR (1994) Efficient fully adaptive wormhole routing in n-dimensional meshes. In: Proceedings of the 14th international conference on distributed computing systems (ICDCS). Pozman, pp 589–596.
12. Feng W, Shin KG (1997) Impact of selection functions on routing algorithm performance in multicomputer networks. In: International Conference on Supercomputing, pp 132–139.
13. Badr HG, Podar S (1989) An optimal shortest-path routing policy for network computers with regular mesh-connected topologies. IEEE Trans Parallel Distrib Syst 38(10):1362–1371
14. Li M, Zeng Q, Jone W (2006) DyXY–a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In: Processing of design automation conference (DAC). San Francisco, pp 849–852.
15. Hu J, Marculescu R (2004) DyAD–Smart routing for network-on-chip. In: Processing of design automation conference (DAC). San Diego, pp 260–263.

16. Dally WJ, Aoki H (1993) Deadlock-free adaptive routing in multicomputer networks using virtual channels. IEEE Trans Parallel Distrib Syst 4(4):466–475
17. Singh A, Dally WJ, Gupta AK, Towles B (2003) GOAL: A load-balanced adaptive routing algorithm for torus networks. In: International Symposium on Computer, Architecture, pp 194–205.
18. Kim J, Park D, Theocharides T, Vijaykrishnan N, Das CR (2005) A low latency router supporting adaptivity for on-chip interconnects. Proceedings of the 42nd annual design automation conference (DAC). ACM, New York, pp 559–564
19. Ascia G (2008), Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. IEEE Trans Comput 57(I.6):809–820.
20. Gratz P, Grot B, Keckler SW (2008) Regional congestion awareness for load balance in networks-on-chip. In: Proceeding of the 14th international symposium on high-performance computer architecture. Salt Lake, City, pp 203–214.
21. Ma S et al. (2011) DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In: Proceeding of 38th annual international symposium on computer architecture (ISCA). San Jose, pp 413–424.
22. Ebrahimi M et al. (2012) CATRA—congestion aware trapezoid-based routing algorithm for on-chip networks. In: Proceeding of design, automation & test in Europe conference & exhibition (DATE). Dresden, pp 320–325.
23. Choi SP, Yeung D-Y (1996) Predictive Q-routing: a memory-based reinforcement learning approach to adaptive traffic control. Adv Neural Inf Process Syst 8(NIPS8):945–951
24. Kumar S, Miikkulainen R (1998) Confidence-based Q-routing: an on-line adaptive network routing algorithm. Smart engineering systems: neural networks, fuzzy logic, data mining, and evolutionary programming 8:147–152
25. Kumar S, Miikkulainen R (1997), Dual reinforcement Q-routing: an on-line adaptive routing algorithm.In: Proceedings of the Artificial Neural Networks in, Engineering Conference, pp 231–238.
26. Feng C, Lu Z, Jantsch A, Li J, Zhang M (2010) A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip. In: Proceedings of NoCArc, pp 11–16.
27. Majer M et al. (2005) Packet routing in dynamically changing networks on chip.In: Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS). Denver, USA.
28. Paliwal KK, George JS, Rameshan N, Laxmi V, Gaur MS, Janyani V, Narasimhan R (2009) Implementation of QoS aware Q-routing algorithm for network-on-chip. In: Contemporary computing. Springer, Berlin, Heidelberg, pp 370–380
29. Puthal MK, Singh V, Gaur MS, Laxmi V (2011) C-routing: an adaptive hierarchical NoC routing methodology. In: IEEE/IFIP 19th international conference on VLSI and system-on-chip (VLSI-SoC). Hong Kong, pp 392–397.
30. Farahnakian F, Ebrahimi M, Daneshtalab M, Liljeberg P, Plosila J (2011) Q-learning based congestion-aware routing algorithm for on-chip network. In: Proceedings of 2nd IEEE international conference on networked embedded systems for enterprise applications (NESEA). Fremantle, pp 1–7.
31. Farahnakian F, Ebrahimi M, Daneshtalab M, Plosila J, Liljeberg P (2012) Adaptive reinforcement learning method for networks-on-chip. In: Proceedings of 12th IEEE international conference on embedded computer systems: architectures, modeling, and simulation (SAMOS XII). Samos, pp 236–243.
32. Ebrahimi M, Daneshtalab M, Farahnakian F, Liljeberg P, Plosila J, Palesi M, Tenhunen H (2012) HARAQ: congestion-aware learning model for highly adaptive routing algorithm in on-chip networks. In: Proceedings of 6th ACM/IEEE International Symposium on Networks-on-Chip (NOCS), pp 19–26.
33. Farahnakian F, Ebrahimi M, Daneshtalab M, Liljeberg P, Plosila J (2014) Bi-LCQ: a low-weight clustering-based Q-learning approach for NoCs. Elsevier J Microprocess Microsyst (MICPRO) 38:64–75
34. Farahnakian F, Ebrahimi M, Daneshtalab M, Liljeberg P, Plosila J (2012) Optimized Q-learning model for distributing traffic in on-chip networks. In: International Conference on Networked Embedded Systems for Enterprise Applications (NESEA), UK, pp 1–8.
35. Varga A et al. (2001) The OMNeT++ discrete event simulation system. In: Proceedings of the European Simulation Multiconference (ESM'2001), pp 319–324.
36. Woo SC et al. (1995) The splash-2 programs: characterization and methodological considerations. In: Proceedings of Computer Architecture (ISCA), pp 24–36.

37. Martin MK, Sorin DJ, Beckmann BM et al (2005) Multifacet's general execution driven multiprocessor simulator (GEMS) toolset. SIGARCH Comput Archit News 33(4):92–99