Adaptive Reinforcement Learning Method for Networks-on-Chip

Fahimeh Farahnakian, Masoumeh Ebrahimi, Masoud Daneshtalab, Juha Plosila, Pasi Liljeberg
Department of Information Technology, University of Turku, Turku, Finland
{fahfar, masebr, masdan, juplos, pakrli}@utu.fi

***Abstract*—In this paper, we propose a congestion-aware routing algorithm based on Dual Reinforcement Q-routing. In this method, local and global congestion information of the network is provided for each router, utilizing learning packets. This information should be dynamically updated according to the changing traffic conditions in the network. For this purpose, a congestion detection method is presented to measure the average of free buffer slots in a specific time interval. This value is compared with maximum and minimum threshold values and based on the comparison result, the learning rate is updated. If the learning rate is a large value, it means the network gets congested and global information is more emphasized than local information. In contrast, local information is more important than global when a router receives few packets in a time interval. Experimental results for different traffic patterns and network loads show that the proposed method improves the network performance compared with the standard Q-routing, DRQ-routing, and Dynamic XY-routing algorithms.**

Keywords; Networks-on-Chip, Adaptive Routing, Dual Reinforcement Learning, Q-routing

I. INTRODUCTION

As technology scales further and chip integration density grows, on-chip communication is playing an increasingly dominant role in System-on-Chip (SoC) design [1][2]. Network-on-Chip (NoC) is a new generation of communication infrastructures for SoC [1][4]. A mesh NoC consists of several cores where each core is connected to a router by a local network interface. Each router is also connected to its neighboring routers through bidirectional links [4][5][6][7]. So, the cores can communicate with each other by propagating packets through routers in the on-chip network.

The performance of NoC strongly depends on the routing techniques. Network routing is the mechanism which allows packets to be forwarded between any pair of source and destination nodes. A routing algorithm should be able to adapt dynamically to the traffic changes in the network. An efficient routing algorithm can alleviate congestion by distributing packets through less congested paths. Generally, routing algorithms can be classified into deterministic and adaptive [8][9][10]. In deterministic routing algorithms [11], a transfer path is completely determined by the source and destination addresses. For example, in the XY routing algorithm, packets first transfer along the X direction, then along the Y direction. In adaptive routing algorithms [9][15], each packet's transfer path determines based on the current network conditions. When network congestion happens, they choose paths with low latencies to avoid congested links and routers.

In our approach, we use a strong method of reinforcement learning —Q-learning. It learns to control a dynamic system optimally through scalar values or rewards. In Q-learning, an agent first percepts the environment and chooses an action. After the action executes, the agent receives a reward. In this way, the agent learns a policy for selecting among actions. Such a policy should maximize the expected sum of discounted rewards. In the other hand, Q-learning allows an agent to learn on-line from experiences, and then by using them, it improves performance [16].

Q-routing [17] is an adaptive routing algorithm which uses the Q-learning [18]. In Q-routing, each node makes routing decisions based on its neighboring nodes information. A node stores a table of Q-values that estimates the quality of alternative paths. These values are updated each time a node sends a packet to one of its neighbors [19]. This way, as the node routes packets, its Q-values gradually incorporate more global information. In DRL method [20], a novel dual reinforcement learning was applied to the satellite communication. This approach is adapted on-line when the system is performing. Dual Reinforcement Q-routing (DRQ-routing) [21] utilizes DRL approach for packet routing. In the DRQ method, each node in the network learns routing policies, results in reducing the average packet delivery time. Learning is performed by carrying the latency information to intermediate nodes (backward exploration unique to DRQ-Routing) and by receiving the learning packets from the neighboring node where a data packet is sent to (forward exploration similar to Q-Routing). At high injection loads, the routing policy learned by DRQ-routing leads to higher performance than Q-routing in terms of average packet delivery time [22].

In this paper, we propose a congestion-aware routing algorithm named Dual Q-routing Adaptive learning Rate (DuQAR). The goal of the presented routing algorithm is to enhance DRQ-routing performance in NoC once the network becomes congested. To make effective routing decisions, the congestion information (Q-values) should be updated continually on the foundation of congestion in the network. Otherwise the routing decision based on unreliable Q-values cannot be accurate. For this purpose, we consider a congestion detection technique which updates the learning rate according to the congestion in each node. The learning rate determines the rate at which newer information overwrites the older one. Therefore, this method adaptively learns an optimal routing strategy and be able to find a less congested path among available paths from the source to destination nodes.

The reminder of this paper is organized as follows. In Section II, the related work is discussed. Some background information on Q-routing and DRQ-routing techniques is given in Section III. In Section IV, the proposed algorithm is explained. The results are reported in Section V, while the summary and conclusion are given in the last section.

II. Related Work

In recent year, significant research has been done to improve the routing efficiency of NoC while most existing adaptive routing techniques are presented to avoid congestion. Adaptive routing policies can be categorized into congestion-oblivious and congestion-aware schemes [23]. In congestion-oblivious algorithms, routing decisions are independent of the congestion condition of the network. Random [24] and Zigzag [25] are two examples of congestion-oblivious methods. In Random routing method, output ports are chosen randomly, while in Zigzag, the output ports are selected based on the remaining hop counts in each dimension. DyXY [26] and DyAD [27] are two approaches of the congestion-aware routing algorithms. These routing algorithms consider the congestion status of the network in the routing decision [28].
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 [9]. DyXY uses local information, the current queue length of the corresponding input port in the neighboring routers, to decide on the next hop. DyXY may lead to forward packets through congested area as employing local information is not sufficient. ANOC [29] was proposed to reduce the network congestion using a cluster-based network. However, it cannot provide global information using the clustering approach.

The Q-routing method is proposed in [17] which is able to find the optimal paths among available paths from the source node to the destination node. This algorithm allows a network to continuously adapt to changing traffic condition by routing packets on routes which estimate the least delivery time. However, Q-routing suffers from the unreliability of the estimated Q-values. The reason is that depending on the traffic pattern and load levels, only few Q-values might be updated regularly while most of the Q-values in the network are unreliable. To address this problem, some other algorithms are presented such as CQ-routing [30] and PQ-routing [31]. In CQ-routing, each Q-value is attached with a confidence value (C-value) which is a measure of how closely the corresponding Q-value represents the current state of the network. PQ-routing keeps track of the last update time and the best Q-values seen so far. PQ-routing is able to explore paths that have been inactive for a long time, and thereby able to restore the previous policy. The same idea is extended to DRQ-routing [21]. The speed of restoration is expected to be higher in DRQ-routing than in the PQ-routing because of the enhanced exploration in DRQ-routing.

Reinforcement learning approaches have rarely been investigated in NoC. The algorithm in [32] is proposed to handle communication among modules which are dynamically placed on a reconfigurable NoC. Another method, named fault-tolerant deflection routing algorithm (FTDR), is presented in [33] which inspired by Q-learning techniques for tolerating faults in NoC. In another approach [34], Q-routing is used to provide the different levels of Quality-of-Service (QoS) such as Best Effort (BE) and Guaranteed Throughput (GT) in NoC. It contrasts the performance of Q-routing with the XY routing strategy in context of QoS. In addition, C-routing [35] is a new cluster-based adaptive routing method which reduces the size of routing tables.

III. Background

This section briefly reviews Q-routing and DRQ-routing techniques.

A. Q-routing

Q-routing first learns a representation of the network state in terms of Q-values and then uses these values to make routing decisions. Each node stores a table of values named Q-values that estimate the quality of the alternative routes. These values are updated each time a node sends a packet to one of its neighbors.

Assume that a message is generated at the source node *s* and it is already at node *x*. This message is going to be sent to the destination node *d* via one of its neighboring nodes, assumed to be *y* (Figure 1). The maximum amount of time it takes for a packet to reach its destination from the node *x* is bounded by the sum of three quantities: (1) the waiting time (q_y) in the input queue of the node *y* (2) the transmission delay (δ) over the link from node *x* to *y*, and (3) the minimum time, $Q_y(z, d)$, it would take for the node *y* to send this packet to the destination via one of the node *y*'s neighbors, assumed to be *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 nodes.

The node *y* sends its best estimate $Q_y(z,d)$ for the destination *d* back to the node *x*. Upon receiving $Q_y(z,d)$, the node *x* computes the new estimate for $Q_x(y,d)$ as follows:

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

$Q_x(y, d)_{est}$ is the node *x*'s best estimated delay that a packet would take to reach its destination node *d* from the node *x* when sent via its neighboring node *y*. So, this value includes the total waiting time and transmission delay over the entire path that it would take starting from the node *y*. Q-value is modified by the following formula after receiving the $Q_x(y, d)_{est}$ value:

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

Learning is performed by updating the Q-values. Learning rate, γ, determines the rate at which newer information overwrites the older one. Learning rate can take a value between zero and one; the value of zero means no learning is made by the algorithm; while the value of one indicates that the most recent information is used.

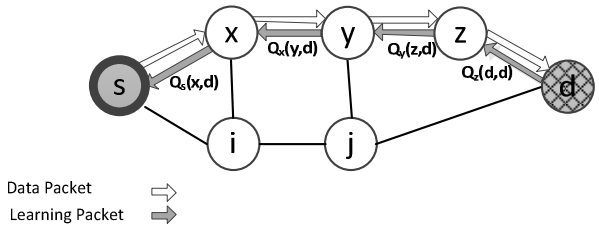

Figure 1. An example of Q-routing method

The Q-routing algorithm has three main steps as follows:

Step 1: Sending a data packet from the node x to the node y

1. *Select a data packet from the queue.*
2. *Find minimum Q-value from Q-table when destination node is d. Assume that, the value belongs to the neighboring node y.*

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

3. *Forward the packet to neighbor y.*

Step 2: Receiving a data packet by the node y

1. *Node y receives a data packet from neighbor x.*
2. *Find the minimum Q-value from the Q-table of node y. Assume that, the neighboring node z has the minimum estimated latency to reach the destination node d.*

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

3. *Measure the waiting time of the packet at the input buffer of node y (q_y) before sending the packet to the neighboring node z.*
4. *Send y's estimate back to node x including $Q_y(z,d)$ and q_y using a learning packet.*

Step3: Receiving a learning packet by the node x

1. *Node x receives the learning packet from node y.*
2. *Extract the estimated Q-value from node y containing $Q_y(z,d)$ and q_y from* the learning packet.
3. *Update the Q-value, $Q_x(y, d)$, regarding the destination node d and neighboring node y using formula (1).*

B. DRQ-routing

Dual Reinforcement Q-routing (DRQ-routing) combines Q-routing with Dual Reinforcement Learning [21]. The Q-routing only updates the Q-value whenever the node receives a learning packet. In other words, when a data packet is sent from the node *x* to the node *y*, the learning packet sends back to the node *x* and thus only the Q-table of the node *x* is updated. Therefore, the Q-routing algorithm only uses forward exploration. The idea of DRQ-routing is to update the Q-tables of both the node *x* and the node *y*. Thereby, the DRQ-routing utilizes the Q-routing for both backward and forward exploration. Forward exploration determines the latency of the remaining path from the current node to the destination node, while backward exploration indicates the latency of the traversed path from the current node to the source node. For an example of the backward exploration consider a case where the node *x* sends a packet to its destination node *d* via its neighboring node *y*. When the data packet is traveling from the source to the destination, it carries some Q-value information between each two neighboring nodes. As the node *y* receives this packet, it uses this information for updating its own estimate of sending a packet to node *s* via node *x*.

As shown in Figure 2, suppose that the packet is currently at the node *x*. This packet contains the minimum latency value from the node *s* to the node *x* passing though the node *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 node *y*, it can update the estimation of sending a packet to the node *s* via neighbor *x*. The new value includes $Q_x(h,s)$ and the waiting time at the input buffer of node *x* (q_x):

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

Where q_x is the waiting time for the packet in the input buffer of the node *x*.

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 nodes. In other words, the receiving node uses the latency information of the traversed path by packets to update the Q-values.

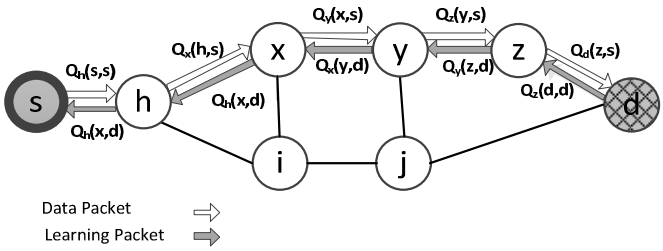

Figure 2. An example of DRQ-routing method

IV. DUAL Q-ROURING ADAPTIVE LEARNING RATE

Dual Q-routing Adaptive learning Rate (DuQAR) provides the ability to adapt the congestion condition of the network in order to alleviate congestion in a NoC. At first the routing table is defined in a way that it could be employed in NoC. Then, the packet header format is modified in order to handle backward exploration. In addition, the learning packet format is defined to be used for carrying information relevant to forward exploration. Finally, the proposed algorithm is presented.

A. Routing table

Each node needs a routing table to maintain information about the routing cost from itself to the possible destination nodes. The table contains *n* entries, where *n* is the number of nodes in the network. As indexed in Table I, each entry has three fields: Next-Router, Latency, and Destination-Router. The Next-Router field determines neighboring nodes which can be used to deliver a packet from the current router to the given destination through shortest paths. The Latency field represents the estimated latency value to deliver a packet from the current node to the destination node via one of the neighboring routers. The Destination field indicates the destination address ID. For example, the contents of Table I

are related to the routing table of the node 5 in the 3×3 mesh network (Figure 3).

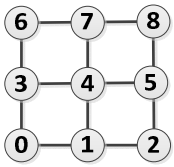

Figure 3. 3×3 mesh architecture

Each entry of the table is allocated to a specific destination in the network that can be reached by the node 5. The latency values are obtained by Formula (1) for forward exploration or by Formula (2) for backward exploration. As shown in the table if there is only one minimal path option, the second field for the Next-Router and the Latency values are set to -1.

Table I. Adjusted Routing Table in NoC

Next-Router		Latency		Destination
2	4	0	0	Node 0
2	4	0	0	Node 1
2	-1	0	-1	Node 2
4	-1	0	-1	Node 3
4	-1	0	-1	Node 4
-	-	-	-	Node 5
4	8	0	0	Node 6
4	8	0	0	Node 7
8	-1	0	-1	Node 8

B. Data Packet and Learning Packet Formats

Two types of packets can be propagated through the network: data packets and learning packets. They use separate virtual channels to propagate information.

The format of the learning packet is illustrated in Figure 4, which consists of four fields as follows:

- *Receiving node ID*: contains the neighboring node ID which is used to forward a learning packet.
- *Forward local latency*: determines the waiting time of a packet in the input buffer before delivering to the next router. In Formula (1), the value of q_y indicates the forward local latency.
- *Forward global latency*: determines the expected latency of a packet from the next node to the destination. In Formula (1), the term $Q_y(z,d)$ indicates the global latency.
- *Destination node ID*: is used for the destination address ID.

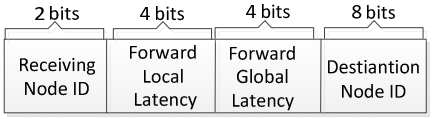

Figure 4. A learning packet format

The header flit format of a typical data packet is shown in Figure 5(a). To support backward exploration, an extra 10-bit is integrated into the header flit as shown in Figure 5(b). The additional fields are described as follows:

- *Sending node ID:* determines the sending node of the data packet. Since each node is connected to at most four neighboring nodes, two bits are enough to encode the neighboring node ID.
- *Backward local latency*: determines the packet's waiting time in the input buffer of the sender node. In the Formula (2) the term q_x represents the local latency. We have assigned four bits to the local latency.
- *Backward global latency*: the expected latency of a packet to propagate from the current node to the source node by passing through the previous node. In Formula (2), the term $Q_x(h,s)$ indicates the backward global latency. We have assigned four bits to this parameter.

Figure 5. Header of the data packet format

C. DuQAR Algorithm

The DuQAR algorithm can be summarized in three steps:

Step 1: Sending a data packet from the node x to node y

1. *Select a packet from the queue.*
2. *Find the minimum estimated latency from the node x to the destination node through one of the neighboring nodes, assumed y:*

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

3. *Find the minimum estimated latency from the node x back to the source node s through the neighboring node h ($Q_x(h,s)$):*

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

Also, find the waiting time of the packet in the input buffer of the node x (q_x). Finally, include these two values into the header of the data packet.
4. *Forward the packet to the neighbor y.*

Step 2: Receiving a data packet by the node y from node x

1. *The node y receives a data packet from the neighbor x*
2. *Backward exploration: Extract the estimation* $(Q_x(h,s)+q_x)$ *from the received packet. Set the learning rate according to the Congestion Detection Method (described in section D) and update the Q-value ($Q_y(x, s)$) using Formula (2).*
3. *Forward exploration: Find the minimum latency estimation from the node y to the destination node d through the neighboring node z:*

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

Also, find the waiting time of the packet in the input buffer of the node y (q_y).
4. *Send the learning packet back to the node x including $Q_y(z,d)$ and q_y.*

Step 3: Receiving a learning packet by the node x from node y

1. *Receive a learning packet from the node y containing $Q_y(z,d)$ and q_y.*
2. *Set the learning rate according to the Congestion Detection Method (described in D) and update the Q-value ($Q_x(y, d)$) using Formula (1)*

D. Congestion Detection Method

The main objective of the DuQAR routing algorithm is to minimize congestion by sending packets through paths with minimum Q-values. Therefore, Q-values should represent the current status of the network. For this purpose, it is necessary to dynamically adapt the Q-values with changing congestion condition of the network; otherwise Q-values are unreliable. If Q-values are frequently updated when the network gets congested, global congestion values from distance nodes become reliable. In contrast, a router may receive few packets in a specific time interval. In this case the values from distance nodes are not accurate and the local values should be more emphasized than the global ones.

Congestion is determined by calculating the average of free buffer slots for each router at a predetermined time interval. The average of free buffer slots is compared with maximum and minimum thresholds at each time interval. If the average of free buffer slots in a router is less than the minimum threshold value, it indicates that the router is not congested and it is unnecessary to update the Q-values regularly. Therefore, the learning rate is set to a minimum value amplifying the impact of local congestion statuses. Moreover, if the average of free buffer slots in a router is greater than the maximum threshold value then the learning rate is set to a large value in order to keep the Q-values as updated as possible. In this way, global information gets more emphasis than local values.

Step 2: Congestion detection method

1. thr_{min}=25% (Total_queue_size);
2. thr_{max}=65% (Total_queue_size);
3. AvgFreeBufferSlots, $count_i$ =0;
4. for t = current simulation time to 200ns OR 100 clk then
5. if $router_i$ received a flit
6. $count_i$=$count_i$+1;
7. $FreeBufferSlots_i$ += $FreeBufferSlots_i[t]$;
8. end if;
9. end for;
10. $AvgFreeBufferSlots_i$ = $FreeBufferSlots_i$/$count_i$;

---

11. if $AvgFreeBufferSlots_i$< =thr_{min}
12. LearnRate= 0.1;
13. else if T_{min} < $AvgFreeBufferSlots_i$<thr_{max}
14. LearnRate = 0.5;
15. else if $AvgFreeBufferSlots_i$>=T_{max}
16. LearnRate= 0.9;
17. End if;

We set the minimum and maximum threshold values to 25% and 65% of the total queue size of a router, respectively. The average of free buffer slots is initialized to zero. The average value is obtained in 200 nanoseconds (100 cycles) time interval (step 4-10) and then the learning rate value will be updated based on the following rules (step 10-17):

Rule1: If the average of free buffer slots is less than the minimum threshold value, then the learning rate is set to *0.1*.

Rule2: If the average of free buffer slots is less than the maximum threshold value and greater than the minimum threshold value, then the learning rate is set to *0.5*.

Rule3: If the average of free buffer slots is greater than the maximum threshold value, then the learning rate is set to *0.9*.

Figure 6 shows an example of the DuQAR algorithm where a packet is transmitted from the source 0 to the destination 4 in a 3×3 mesh network. Suppose that the congestion conditions in the node 0, node 1, and node 4 are medium, low, and high, respectively. At first, the node 0 selects the path with minimum Q-value belonging to one of the neighboring nodes 1 and 3. Suppose that the Q-value in the node 1 is smaller than the Q-value in the node 3. Therefore, the node 0 sends the data packet to the node 1. As shown in Figure 6(a), the local latency in the header flit of the data packet is set to q_0 indicating the waiting time of the packet in the input buffer of node 0. The global latency is equal to zero because the node 0 is the source node. When the intermediate node 1 receives the data packet from the node 0, backward exploration information is extracted from the data packet. Based on this information, the corresponding row of the routing table in the node 1is updated. Since the congestion condition of router 1 is low, the Rule1 is satisfied and the learning rate is set to *0.1*. This learning rate is used in updating the Q-values. Therefore, according to Formula (2), the new Q-value is:

$$Q_1(0,0)_{new} = Q_1(0,0)_{old} + 0.1(0 + q_0 + 0 - Q_1(0,0)_{old})$$

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

As can be seen from Figure 6(b), the node 1 not only sends the data packet to the destination node but also generates a learning packet and sends it back to the node 0. As the node 0 receives the learning packet, it updates the Q-value related to the destination node 4 using the Formula (1). Since the average of free buffer slots value is between the maximum and minimum threshold in the last time interval, the learning rate γ is set to *0.5*.

$$Q_0(1,4)_{new} = Q_0(1,4)_{old} + 0.5(Q_1(4,4) + q_1 + \delta - Q_0(1,4)_{old})$$

The $Q_1(4,4)$ value is equal to the link delay (assumed to be zero) because a packet can reach its neighboring node in one hop. The node 4 uses the latency value in the data packet. This value is essentially an estimation of the minimum time it would take for the packet to reach back to its source 0 from the node 1.

Upon receiving this estimation, the node 4 computes the new estimation for $Q_4(1, 0)$ as follows:

$$Q_4(1,0)_{new} = Q_4(1,0)_{old} + 0.9(Q_1(0,0) + q_1 + 0 - Q_4(1,0)_{old})$$

Finally, the Q-value in the node 1 is updated by receiving the learning packet from the node 4 (as can be seen in Figure 6(c)). The node 1 extracts forward exploration information from learning packet. The $Q_1(4,4)_{new}$ is computed as:

$$Q_1(4,4)_{new} = Q_1(4,4)_{old} + 0.5(0 + q_4 + \delta - Q_1(4,4)_{old})$$

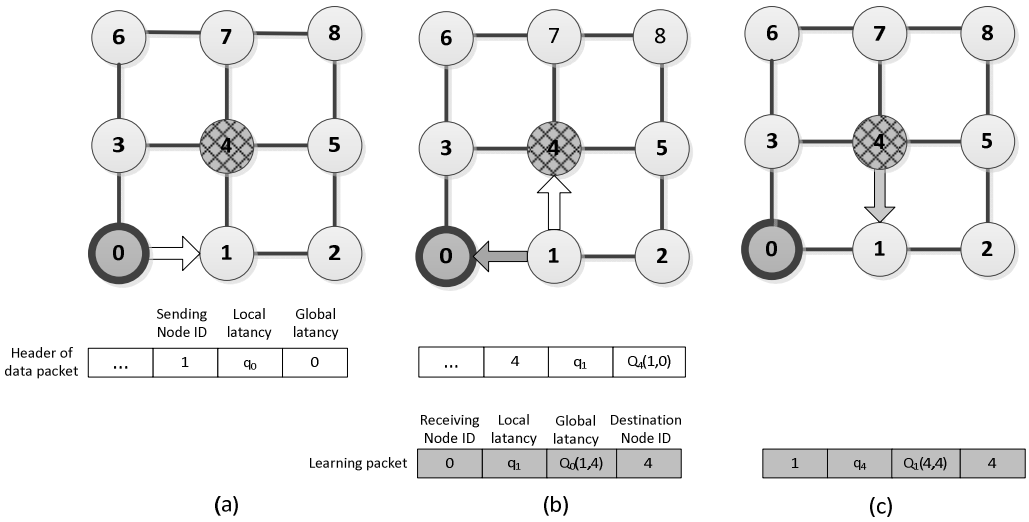

Figure 6. DuQAR algorithm for a 3×3 mesh

V. EXPERIMENTAL RESULT

To evaluate the efficiency of our approach, implementations have been performed on the OMNET framework [36]. OMNeT++ is an extensible, modular, open-source component-based C++ simulation library and framework, primarily aimed at building network simulators [37]. The proposed approach has been compared with three other algorithms, the standard Q-routing, DQ-routing, and DyXY-routing. Two dimensional mesh configurations have been used for the NoC. The simulator inputs include the network size, the network offered load, the routing algorithm, and the traffic type. The amount of packets injected into the network depends on the value of the network offered load. Below is the Formula for calculating the network offered load:

Network offered load = flit_size/flit-arrival-delay

flit_size represents the size of the flit. It is assumed that the data packets have a fixed length of 8 flits with the flit width of 32 bits. *Flit-arrival-delay* represents the delay time between a previous flit generation and next flit generation.

The simulations were conducted on a 4×4 mesh under various traffic patterns. For each simulation, packet latencies are averaged over 10,000 packets. To propagate data packets, two virtual channels are used along the *x* and *y* dimensions, while a separate virtual channel is allocated to learning packets. The buffer size of first two virtual channels is set to eight flits. Since the network performance is greatly influenced by the traffic pattern, we applied three traffic patterns, namely uniform random, hotspot and transpose.

A. Uniform Random Traffic Profile

In the random traffic profile, each core sends a packet to another core with a random probability. The destination of different packets in each router is determined randomly using a uniform distribution. Figure 7 shows the average communication latency as a function of the average packet injection rate. As it can be observed from the results, DuQAR learns an effective policy nearly as fast as Q-routing and DRQ-routing in low traffic loads. DuQAR routing algorithm leads to a lower latency than the other algorithms in medium and high traffic loads.

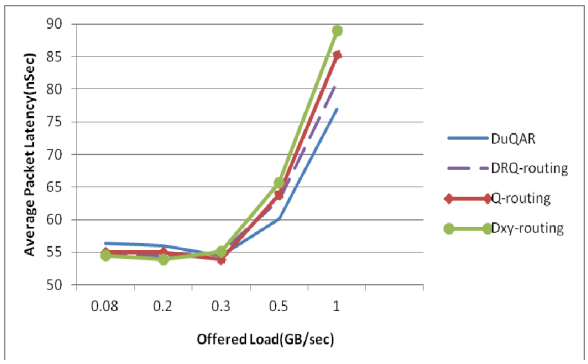

Figure 7. Average packet latency under uniform random traffic

B. Transpose Traffic Profile

In this traffic pattern, a node (i,j) only sends a message to the node (j,i). Figure 8 shows the performance of four routing algorithms under the transpose traffic. It can be seen that DuQAR and DQ-routing still have better performance when network congestion happens. DuQAR performs the best when the packet injection rate increases (offered load >1GB/sec).

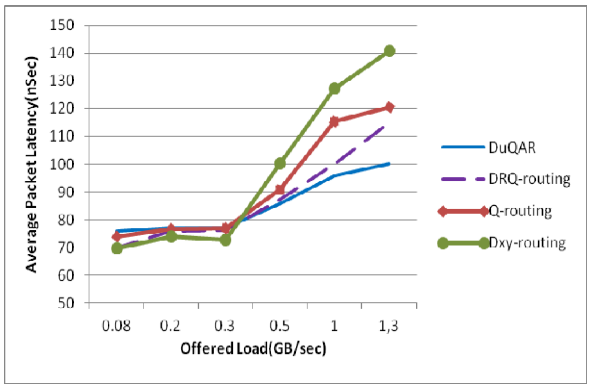

Figure 8. Average packet latency under transpose traffic

C. *Hotspot Traffic Profile*

Under the hotspot traffic pattern, one or more nodes are chosen as hotspots receiving an extra portion of the traffic in addition to the regular uniform traffic. In simulations, given a hotspot percentage of *H*, a newly generated message is directed to each hotspot node with an additional *H* percent probability. We simulate the hotspot traffic with a single hotspot at node 9 in the 4×4 2D-mesh network. The average packet latency of each network with *H=10%* are illustrated in Figure 9. As observed from the results, Q-routing algorithm has lowest average latency at low traffic loads while DuQAR performs the best in medium and high traffic loads.

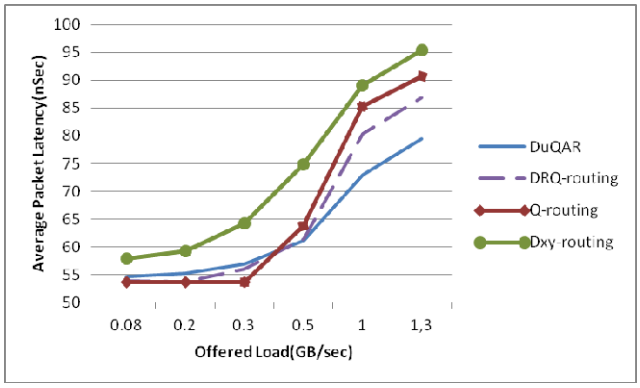

Figure 9. Average packet latency under hotspot traffic

Figure 10 and Figure 11 show the Q-values (latency) variations of routers 9 and 6 under the hotspot traffic. The variation of Q-values is plotted on the vertical axis and simulation time on the horizontal axis. The values on vertical axis can be seen as the latency when the node 6 or node 9 receives a packet. We consider the node 9 as a hotspot and node 6 as a low congested node.

The line on the chart displays the average Q-values at the simulation time. Experimental results show that during the learning process, the average Q-values in the node 6 are often much less than the average Q-values in the node 9. This is because at node 9, the learning is more happening and clearly demonstrates that the changing of the Q-values is very high in the node 9. After 80000 nanoseconds, the variations of Q-value reduced to around 50% from what it was at the beginning of the learning process.

Table II illustrates the performance gain of our proposed method over Q-routing, DRQ-routing, and Dynamic XY-routing algorithms near the saturation point (0.5).

Table II. Performance gain for three traffic patterns

Traffic Pattern	DyXY-routing	Q-routing	DQ-routing
Uniform	8.3%	5.7%	4.6%
Transpose	14.2%	5.2%	2.4%
Hotspot	18.3%	4.6%	3.5%

D. *Hardware Cost*

The hardware cost of our proposed method along with Q-routing, DRQ-routing, and Dynamic XY-routing schemes is

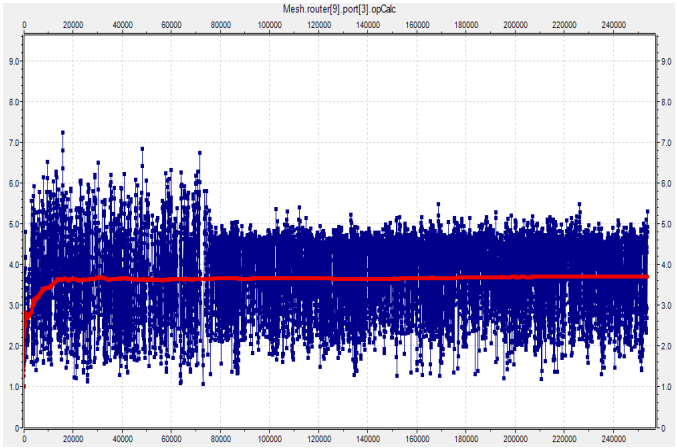

Figure 10.learning curve of the router 9 under hotspot traffic by DuQAR method

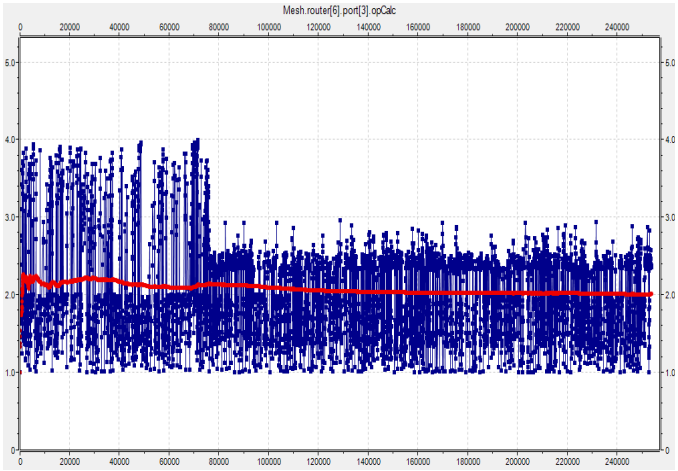

Figure 11. Learning curve of the router 6 under hotspot traffic by DuQAR method

measured. For this purpose, the on-chip router of each scheme is implemented with VHDL and synthesized with Synopsys Design Compiler using the 65nm standard CMOS technology with a timing constraint of *1GHz* for the system clock and supply voltage of *1V*. The synthesized netlist is verified through post synthesis simulations. The layout areas of the four schemes are listed in Table III. The area overhead of DuQAR is comparable with the other learning methods.

Table III.Hardware cost

Method	Router Area (mm^2)
DuQAR –routing method	0.1705
Dual Q-routing method	0.1689
Q-routing method	0.1683
DyXY method	0.1503

VI. CONCLUSION

In this paper, we presented a congestion-aware adaptive routing algorithm (DuQAR algorithm) for on-chip networks. The proposed method provides the routing policy to alleviate

congestion in the network by estimating latency values between each pair of source and destination nodes of the network. For this purpose, we considered a congestion detection method that calculates the average of free buffer slots in each time interval. Then, this value is compared with maximum and minimum thresholds to determine congestion level in the router. If the router is congested, then the latency value must be frequently updated. So, the learning rate is set to a large value in order to keep the global latency values as updated as possible. On the other hand, local information is in more emphasized than global values when a router is not congested. The experiments show that DuQAR routing method is able to route packets more efficiently than Q-routing, DQ-routing, and Dxy-routing in medium and high network loads with a small hardware overhead.

REFERENCES

[1] D. Wu, B. M. Al-Hashimi, M. T. Schmitz, "Improving Routing Efficiency for Network-on-Chip through Contention-Aware Input Selection", in Proc. of 11th Asia and South Pacific Design Automation Conference, pp. 36-41, 2006.

[2] T. C. Xu et al., "A Minimal Average Accessing Time Scheduler for Multicore Processors," In Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), pp.287-299, 24-26 October 2011, Australia.

[3] M. Daneshtalab et al., "Adaptive Input-output Selection Based On-Chip Router Architecture," Journal of Low Power Electronics (JOLPE), Vol. 8, No. 1, pp. 11-29, 2012.

[4] L. Benini, G. De Micheli, "Networks on chips: A new SoC paradigm", Computer, pp. 70-78, 2002.

[5] J. Henkel, W. Wolf, and S. Chakradhar, "On-chip networks: A scalable, communication-centric embedded system design paradigm", VLSI Design,pp. 845-851,2004.

[6] T. C. Xu et al., "A Study of 3D Network-on-Chip Design for Data Parallel H.264 Coding," Journal of Microprocessors and Microsystems, Vol. 35, No. 7, pp. 603-612, October 2011.

[7] M. Daneshtalab et al., "Memory-Efficient On-Chip Network with Adaptive Interfaces," IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems (IEEE-TCAD), Vol. 31, No. 1, pp. 146-159, Jan 2012.

[8] L. M. Ni, P. K. McKinley, "A survey of wormhole routing techniques in direct networks", Computer, pp. 62-76,1993.

[9] M. Ebrahimi et al., "CATRA-Congestion Aware Trapezoid-based Routing Algorithm for On-Chip Networks," in Proceedings of 15th ACM/IEEE Design, Automation, and Test in Europe (DATE), pp. 320-325, Mar. 2012, Germany.

[10] F. Farahnakian et al., "Q-learning based Congestion-aware Routing Algorithm for On-Chip Network," in Proceedings of 2th IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA), pp. 1-7, Dec. 2011, Australia.

[11] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. Van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip", IEE Proceedings: Computers and Digital Techniques, pp. 294-302, 2003.

[12] E. Nilsson, M. Millberg, J. Oberg and A. Jantsch, "Load distribution with the proximity congestion awareness in a network on chip", DATE, pp. 1126-1127, 2003.

[13] T. T. Ye, L. Benini, G. De Micheli, "Packetization and routing analysis of on-chip multiprocessor networks", Journal of Systems Architecture, pp. 81-104, 2004.

[14] M. Daneshtalab et al., "NoC Hot Spot minimization Using AntNet Dynamic Routing Algorithm," in Proceedings of 17th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 33-38, Sep 2006, USA.

[15] M. Dehyadegari et al., "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, June 2011, USA.

[16] R.S. Sutton and A.G. Barto, "Reinforcement Learning. An Introduction", MIT Press, Cambridge, MA, 2000.

[17] J.A.Boyan, M. L .Littman, "Packet routing in dynamically changing networks:A reinforcement learning approach", Advances in Neural Information Processing Systems, pp. 671-678,1994.

[18] C.J.C.H. Watkins and P. Dayan, "Q-Learning", in Proc. Machine Learning, pp.279-292, 1992.

[19] J. Boyan and M. Littman, " A Distributed Reinforcement Learning Scheme for Network Routing", Technical report, Department of Computer Science, Carnegie Mellon University,1993.

[20] P. Goetz, S. Kumar, R. Miikkulainen, "On-Line Adaptation of a Signal Predistorter through Dual Reinforcement Learning," Proc. machine Learning: Proceedings of the 13th Annual Conference ,1996.

[21] S. Kumar and R. Miikkulainen, "Dual reinforcement Q-routing: An on-line adaptive routing algorithm", in Proc. of the Artificial Neural Networks in Engineering Conference, pp. 231-238, 1997.

[22] S.Kumar, "Confidence based Dual Reinforcement Q-routing: an On-line Adaptive Network Routing Algorithm", Master's thesis, Department of Computer Sciences, In the University of Texas at Austin. pp. I98-267, 1998.

[23] P. Gratz, B. Grot and S.W. Keckler, "Regional Congestion Awareness for Load Balance in Networks-on-Chip", in Proc. of HPCA, pp. 203–214, 2008.

[24] H.G. Badr, S. Podar, "An optimal shortest-path routing policy for network computers with regular mesh-connected topologies", pp.1362-1371, 1989.

[25] W. Feng and K. G. Shin, "Impact of Selection Functions on Routing Algorithm Performance in Multicomputer Networks", In International Conference on Supercomputing, pp. 132–139, 1997.

[26] M. Li, Q. Zeng, W. Jone, "DyXY - a proximity congestion-aware deadlock-free dynamic routing method for network on chip", in Proc. of DAC, pp. 849-852, 2006.

[27] 4. J.C. Hu and R. Marculescu, "DyAD – Smart Routing for Network-on-Chip", Design and Automation Conference, 2004.

[28] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, C. R. Das, "A Low Latency Router Supporting Adaptivity for On-Chip Interconnects", in Proc. of DAC, pp. 559-564, 2005.

[29] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, and H. Tenhunen, "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, Oct 2011.

[30] S. Kumar and R. Miikkualainen, " Confidence-based Q-routing: an on-queue adaptive routing algorithm", In Proceedings of Neural Networks in Engineering, 1998.

[31] Choi, S. P. M., Yeung, D.-Y., "Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Trac Control", In Advances in Neural Information Processing Systems 8, pp. 945{951. MIT Press, Cambridge, MA,1996.

[32] J.A.Boyan, M. L .Littman, "Packet routing in dynamically changing networks:A reinforcement learning approach", Advances in Neural Information Processing Systems 6., pp. 671-678,1994.

[33] C. Feng, Z. Lu, A. Jantsch, j. Li, and M. Zhang, "A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip", in Proc of NoCArc, pp.11-16 , 2010.

[34] K. K. Paliwal, J. S.George, N. Rameshan, V. Laxmi, M.S.Gaur, V. Janyani, and R.Narasimhan, "Implementation of QoS Aware Q-Routing Algorithm for Network-on-Chip", pp. 370-380, 1C3 2009.

[35] M.K. Puthal, V. Singh, M.S. Gaur,and V. Laxmi, "C-Routing: An Adaptive Hierarchical NoC Routing Methodology" , 19th International Conference on VLSI and System-on-Chip, pp. 392 – 397, 2011.

[36] A. Varga et al., "The OMNeT++ discrete event simulation system", In Proc. of the European Simulation Multiconference (ESM'2001), pp. 319-324, 2001.

[37] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny, "NoCs simulation framework for OMNeT++", in Proc. of NOCS, pp.265-266, 2011.