# Q-learning based Congestion-aware Routing Algorithm for On-Chip Network

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

*Abstract*— **Network congestion can limit performance of NoC due to increased transmission latency and power consumption. Congestion-aware adaptive routing can greatly improve the network performance by balancing the traffic load over the network. In this paper, we present a reinforcement learning method, Q-learning, for NoC to alleviate congestion in the network. In the proposed method, local and nonlocal congestion information is propagated over the network utilizing learning packets. This learning approach results in better routing decisions due to up-to-date and more reliable congestion values. According to this congestion information, a path is chosen for a packet which is less congested. Experimental results with synthetic test cases demonstrate that the on-chip network utilizing the proposed method outperforms a conventional scheme, Dynamic XY, (28% for uniform traffic and 17% for hotspot traffic) with a 12% of area overhead.**

***Keywords: Networks-on-Chip, Congestion-aware Adaptive Routing, Reinforcement Learning, Q-routing***

## I. INTRODUCTION

As technology scales and chip integration grows, on-chip communication is playing an increasingly dominant role in System-on-Chip (SoC) design [1]. Networks-on-chip (NoCs) have emerged as a new communication infrastructure to address scalability, reliability and throughput [2][3][4][5][6][7]. In NoC architecture, Processing Elements (PEs) are connected to the routers, so that the PEs can communicate with each other by propagating packets through the on-chip network. Routing algorithms are used in NoCs in order to send a packet *P(s, d)* from a source node *s* to a destination node *d*. The performance of NoC depends strongly on the routing techniques. Routing algorithms are classified as deterministic or adaptive. Deterministic routing algorithms always provide a fixed path between the given source and destination pair. These algorithms are unable to alleviate congestion since the routing decision is independent of the network condition. However, congestion is a major factor in limiting performance of NoC and needs to be avoided [8]. Adaptive routing algorithms allow messages to choose among multiple paths between source and destination. Therefore, they can decrease the probability of passing packets through congested areas and adapt the congestion condition to the network status[9].

Reinforcement learning is a methodology to obtain an optimal solution when the system offers alternative choices. It has attracted considerable attention in the past because it provides an effective approach for problems in which optimal solutions are analytically unavailable or difficult to obtain. The learning methodology is based on the common-sense idea that if an action is followed by a satisfactory state, or by an improvement, then the tendency to produce that action is strengthened, i.e., reinforced. On the other hand, if the state becomes unsatisfactory, then that particular action should be suitably punished [10][11].

Q-learning [12] is one of the algorithms in the reinforcement learning family of machine learning. In the Q-learning approach, the learning agent first learns a model of the environment on-line and then utilizes this knowledge to find an effective control policy for the given task.

Q-routing [28] is a network routing method based on the Q-learning algorithm. Each node makes its routing decisions based on the information on their neighboring nodes. A node stores a table of Q-values that estimate the quality of the alternative paths. These values are updated each time the node sends a packet to one of its neighbors. This way, as the node routes packets, its Q-values gradually incorporate more global information. Such exploration has proven to be capable of adapting to load changes and to perform better than the deterministic routing with high loads [13].

The main contribution of this paper is to apply and utilize the Q-learning method to evaluate local and global congestion information in NoCs. In the proposed approach, named Q-learning based Congestion-aware Algorithm (QCA), whenever a packet is sent from node X to node Y, a learning packet returns the local and nonlocal congestion information back to node X. The local congestion information indicates the waiting time of a packet at the corresponding input buffer of node Y. The nonlocal congestion information gives a global view of the congestion conditions of the nodes located between node Y and the destination node. This information is extracted from a Q-table available at each router and is updated using learning packets. The aim of QCA method is to estimate and predict the congestion condition of the network as close to the actual values as possible. This information is used in the routing decision to choose a less congested path.

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

## II. RELATED WORK

2D-mesh topology is a popular architecture for NoC design because of its simple structure, ease of implementation, and support for reuse [17]. The performance and efficiency of NoCs largely depend on the underlying routing methodology. Adaptive routing algorithms can be decomposed into routing and selection functions. The routing function supplies a set of output channels based on the current and destination nodes. The selection function selects an output channel from the set of channels supplied by the routing function [18]. The selection functions can be classified as either congestion-oblivious or congestion-aware schemes [19]. In congestion-oblivious algorithms, such as Zigzag [20] and random [21], routing decisions are independent of the congestion condition of the network. This policy may disrupt the load balance since the network status is not considered. On the other hand, congestion-aware routing algorithms consider the congestion status of the network in the routing decision [9]. Several methods have been presented in order to address the network congestion problem [15][16][22]. In DyXY [23] the selection is performed using the local congestion status of the network [18]. In this scheme, based on the static XY algorithm, 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 routers to decide on the next hop. It is assumed that the collection of these local decisions should lead to a near-optimal path from the source to the destination. The main weakness of DyXY is in the use of the local information in the routing decision that may lead to forward packets through congested area as employing local information is not sufficient. ANOC, presented in [22], was proposed to reduce the network congestion using a cluster-based network which imposes additional hardware overhead.

Most of the other congestion-aware algorithms [14][15], also consider local traffic condition. Routing decisions based on local congestion information may lead to an unbalanced distribution of traffic load. Therefore, they are efficient when the traffic is mostly local, i.e. cores communicate with other ones close to them [24], but they are unable to solve the global load balance problem via making local decisions.

Q-learning based approaches have been rarely investigated in NoCs. The algorithm in [28] is proposed to handle communication among modules which are dynamically placed on a reconfigurable NoCs. Another method, named fault-tolerant deflection routing algorithm (FTDR), is presented in [25] which inspired by Q-learning techniques for tolerating faults in NoC.

## III. PRELIMINARILY AND BACKGROUND

In this section, we review the preliminaries about Q-learning and Q-routing techniques.

### A. *Q-learning*

One of the most important breakthroughs in reinforcement learning is known as Q-learning. Q-learning algorithms use a table to estimate Q-values, called Q-table. As shown in Table I, each table maintains four fields named Current-State Space, Next-State Space, Action Space and Q-value. Current-State Space refers to everything that the agent currently perceives, while Next-State Space is determined by the Current-State and the actions selected by the agents. The Action space indicates the actions that the agent can perform. Each Q-value is accessible by $Q$(s, a) representing the expected reinforcement of taking action $a$ in state $s$.

Table I. Q-table

| Current-State Space | Next-State Space | Action Space | Q-value |
|---|---|---|---|
| . | . | . | . |
| . | . | . | . |

The standard Q-learning algorithm has several stages as follows [26]:

**Q-learning algorithm**

```
1. For each $s$ and $a$, initialize the table entry $Q(s, a)$ to zero
2. Observe the current state $s$
3. Select action $a$ through one of these methods and execute it:
     • Exploration or random
     • Exploitation or Q-table based
4. Receive reaction $r$
5. Observe the new state $s'$ and update the table entry for $Q(s, a)$
   using the values of $r$ and $Q(s', a)$
6. $s \leftarrow s'$
7. Go back to step1
```

At each time, an agent is assumed to be in a certain state $s$, and it chooses action $a$ to interact with the environment. Then the agent receives a reward $r$ from performing action $a$ and observes a new state $s'$. After that, Q-table is updated.

There are two methods for selecting action $a$ from the possible actions in every state [11]:

*Exploration or random action selection:* optimal actions, which are not chosen yet, are added to the table.

*Exploitation or Q-table based*: actions are selected according to the learned Q-table.

It is clear that action selection is more exploration at the beginning of learning, and is more exploitation towards the end of learning.

## B. *Q-routing*

The task of Q-routing is to find an optimal routing policy for the network [27]. In Q-routing, Q-learning is used to first learn a representation of the network state in terms of Q-values and then use these values to make routing decisions. Each node $x$ in the network represents its own view of the network state through its Q-table. Given this representation of the state, the action $a$ at node $x$ is to find a neighboring router to deliver a packet which results to lower latency for the packet to reach its destination.

As soon as node $x$ sends a packet, destined for node $d$, to one of its neighboring nodes $y$, node $y$ sends its best estimate $Q_y(z,d)$ for the destination $d$ back to node $x$:

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

Where N(y) is a set of the y's neighboring nodes.

This value essentially estimates the remaining time in the journey of packet *P (s; d)*. Upon receiving $Q_y(z,d)$ node $x$ computes the new estimate for $Q_x(y,d)$ as follows:

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

$Q_x(y;\ d)$ is node $x$'s best estimated delay that a packet would take to reach its destination node $d$ from node $x$ when sent via its neighboring node $y$ excluding any time that this packet would spend in node $x$'s queue, and including the total waiting time and transmission delay over the entire path that it would take starting from node $y$. Q-value is modified by the following formula:

$$Q_x(y,d)_{new} = Q_x(y,d)_{old} + \gamma\left(Q_y(z,d) + q_y + \delta - Q_x(y,d)_{old}\right) \qquad (1)$$

Therefore, the maximum amount of time it will take for a packet to reach its destination from node $x$ is bound by the sum of three quantities: (1) the waiting time ($q_y$) in the packet queue of node $y$ (2) the transmission delay ($\delta$) over the link from node $x$ to $y$, and (3) the time $Q_y(z;\ d)$ it would take for node $y$ to send this packet to its destination via any of node $y$'s neighbors ($z$). Moreover, the learning rate, $\gamma$, determines the rate at which newer information overwrites the older one. $\gamma$ 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 only the most recent information is used.

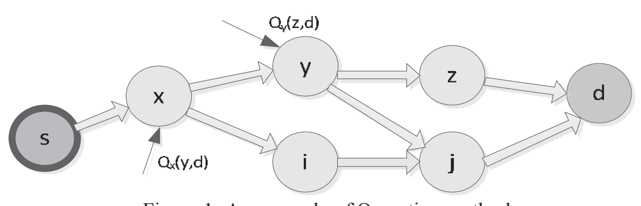

Figure 1. An example of Q-routing method

## IV. Q-LEARNING BASED CONGESTION-AWARE ALGORITHM (QCA)

In this section, in order to utilize the Q-routing technique in NoC, at first we modify Q-table in a way that it could be used in NoCs. Then we explain the format of the learning packets, carrying the information relevant to the Q-routing. Finally, we describe the proposed Q-learning algorithm in NoCs.

### A. *NoC-based adjusted Q-table*

Q-table should be adapted in order to be applicable in NoCs. An adapted Q-table can be considered as a $n\times m$ matrix, where $n$ is the number of nodes in the network while $m$ is the number of columns. As indexed in Table II, the column of the table has five fields: Current-Router, Next-Router, Output-Port, Latency, and Destination-Router. The Current-Router and Next-Router fields are referred to current state space and next state space in Q-table (Figure 1), respectively. The Output-Port indicates the action space that can be within the following cases: north, south, west, east, and local directions. The Latency is representative of the estimated latency value to deliver a packet from the current to the destination node via the next router.

For example, the contents of Table II are related to the adapted Q-table of node 4 in 3×3 mesh network (Figure 3). Each entry of the table is allocated to a specific destination in the network that can be reached by node 4. The Output-Port field indicates the minimal directions which can be used to deliver the packet from node 4 to the given destination; while the Next-Router field points to the neighboring nodes in those directions. The latency value is obtained by formula (1).

Table II. Adjusted Q-Table in NoC

| State-Space | | | Action-Space | | Q-Value | | Goal |
|---|---|---|---|---|---|---|---|
| s | s' | | a | | Q(s; a) | | d |
| Current-Router | Next-Router | | Output-Port | | Latency | | Destinatio n- Router |
| Node 4 | 1 | 3 | South | West | | | Node 0 |
| Node 4 | 1 | - | South | - | | - | Node 1 |
| Node 4 | 1 | 5 | South | East | | | Node 2 |
| Node 4 | 3 | - | West | - | | - | Node 3 |
| Node 4 | 4 | - | Local | - | - | - | Node 4 |
| Node 4 | 5 | - | East | - | | - | Node 5 |
| Node 4 | 3 | 7 | West | North | | | Node 6 |
| Node 4 | 7 | - | North | - | | - | Node 7 |
| Node 4 | 5 | 7 | East | North | | | Node 8 |

### *B. Learning Packet Format*

Two types of packets can be propagated through the network, data packets and learning packets. However, they use separate virtual channels to propagate information. As shown in Figure 2, the learning packet is a 1-flit packet consisted of four main fields described as follows:

- *Receiving node ID*: determines the corresponding neighboring node to forward a learning packet. Since each node is connected to at most four neighboring nodes, 2 bits is enough to encode the neighboring nodes ID.
- *Local latency*: the packet's waiting time in the input buffer of the neighboring router to receive an output channel. In the formula (1) the term $q_y$ is referred to local latency. We have considered 2 bits for the local latency.
- *Global latency*: the latency that a packet probably experience from the next node to the destination. In formula (1), the term $Q_y(z,d)$ indicates the global latency. We have considered 4 bits for the global latency.
- *Destination node ID*: determines the destination node of a data packet. 8 bits are considered for this purpose, so that it is sufficient for 16×16 mesh network. For bigger size network, more bits should be assigned to destination ID field.

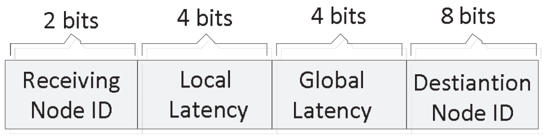

Figure 2. A learning packet format

### *C. The congestion-aware routing algorithm based on Q-routing approach*

Suppose that packet *p* is sent from the source node *s* to destination *d*. At each router, the routing unit decides where to send a packet. The decision is made based on the congestion values received from the neighboring nodes. The *minimum selection function* selects a path which is less congested. When the current router sends a data packet to the downstream router, it is also required to return the learning packet (i.e. containing local and global congestion information) back to the upstream router. Upon receiving the learning packet, the upstream router updates the corresponding entry in Q-table using the *update Q-table function*. The proposed algorithm along with the *minimum selection* and *Q-table functions* are given as follow:

**The Proposed Algorithm**

```
1. A new packet is delivered from node x to node y.
2. At node y, the minimum selection function returns the neighbour
   of node y with the smallest congestion value.
3. The packet is sent from the current node y to the downstream
   router; concurrently a learning packet is delivered to node x.
4. Node x receives the learning packet and updates Q-value in the
   corresponding row of Q-table, calling the update Q-table
   function.
```

In the minimum selection function, when node *x* receives a packet destined for node *d*, it checks the vector $Q_x(*; d)$ of Q-values to select a neighboring node *y* in which the $Q_x(y; d)$ value is minimum (note that * determines all neighboring nodes of node x in minimal directions). It is important to note that these Q-values are not exact. They estimate the packet latency from the current to the destination node and the routing decision based on these estimates does not necessarily give the best solution.

**Minimum Selection Function**

```
1.  Dest_id ← Get_Destination_Router();  --From the packet header
2.    if destination is the local router then
3.        consume the packet
4.    end if;
5.    if number of neighboring nodes=1 then
6.        Selected-Router ← Q-table(Dest_id)(Next_Router1);
7.    end if;
8.    if number of neighboring nodes=2 then
9.         Latency_Neighbor1 ← Q-table(Dest_id)(Latency_Path1)
10.        Latency_Neighbor2 ← Q-table(Dest_id)(Latency_Path2)
11.        if Latency_Neighbor1<= Latency_Neighbor2 then
12.            Selected-Router ← Q-table(Dest_id)(Next_Router1);
13.        else
14.            Selected-Router ← Q-table(Dest_id)(Next_Router2);
15.        end if;
16.  end if;
```

When a learning packet reaches a router (i.e. node *x*), Q-table should be updated in order to be adapted to the changing state of the network. We suppose that the link delay, *δ*, is a constant value between each two adjacent nodes, so in formula (1) we set it to the value of zero. Moreover, a 50-50 weight is assigned to old and new information. Thus simple weighting assignment has performed the best compared to other cases, so that a value of 0.5 is assigned to γ. Therefore, the formula (1) is rewritten as:

$$Q_x(y,d)_{new} = Q_x(y,d)_{old} + 0.5\big(Q_y(z,d) + q_y - Q_x(y,d)_{old}\big)$$

In other words, each router learns a more accurate model of the dynamically changing network condition.

**Updates Q-table Function**

```
For each router which received learning packet
1.  --extracted from the learning packet header
2.  Node_id ← get_Receicing_Node_id()
3.  Local_Latency ← get_Local_Latency()
4.  Global_Latency ← get_Global_Latency()
5.  Dest_id ← get_Destination_Node_id()
6.  -----------------------------------------------------
7.  if Node_id= Q-table(dest_id)(Next_Router1) then
8.      Q_value_old ← Q-table(dest_id)(Latency_Path1)
9.      Q-table(dest_id)(Latency_Path1) ← Q_value_old+
        0.5(Global_Latency+Local_Latency-Q_value_old)
10. end if;
11. if Node_id= Q-table(dest_id)(Next_Router2) then
12.     Q_value_old ← Q-table(dest_id)(Latency_Path2)
13.     Q-table(dest_id)(Latency_Path2) ← Q_value_old+
        0.5(Global_Latency+Local_Latency-Q_value_old)
14. end if;
```

Figure 3 shows an example where a data packet is going to be sent from node 0 to the destination 5. Whenever, the data arrives at the intermediate node 1, the routing unit at node 1 decides either to send a packet to node 2 or node 4. Suppose the routing unit selects node 2 for the next hop which is less congested. At this time, a learning packet is generated and delivered back to node 0. This learning packet should aggregate the local and global congestion information. To obtain the local congestion information, the waiting time at the input buffer of node 1 is considered; while for the global information, the latency value in Q-table is utilized. This information is extracted from the corresponding row of Q-table at node 1 indicating the estimated latency required to send a packet from node 1 to the destination node 5 through node 2.

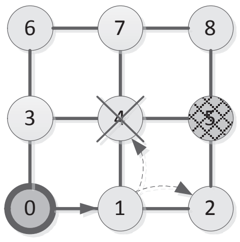

Figure 3. 3×3 mesh network

## V. EXPERIMENTAL RESULT

In this paper we used an OMNeT++ based NoC simulator to calculate the average delay and the throughput. OMNeT++ is an extensible, modular, open-source component-based C++ simulation library and framework, primarily aimed at building network simulators [29][30]. A two dimensional mesh configuration has been used for the NoC. The simulator inputs include the network size, the network offered load, the routing algorithm, and the traffic type. Experiments are conducted to evaluate the performance of QCA method compared to the DyXY routing algorithm. Simulations are carried out on a 4×4 mesh NoC. The performance of the routing scheme is evaluated through latency-throughput curves. For a given packet injection rate, a simulation is conducted to evaluate the average packet latency. It is assumed that packet 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. For each simulation, the packet latencies are averaged over 5000 packets. It is assumed that the data packets have a fixed length of 8 flits with the flit width of 32 bits; and the learning packets are 1-flit packets with the flit width of 16 bits. To propagate data packets, two virtual channels are used along the y and x dimensions, while for learning packets, one virtual channel is utilized. The buffer size per virtual channel is set to 4 flits. Since the network performance is greatly influenced by the traffic pattern, we applied two different traffic patterns, including uniform random and hotspot. In each simulation time step (nanosecond), these packets are stored in the queue of the source nodes. The amount of injected packets depends on the value of the network offered load. Below is the formula for calculating the network offered load:

$$\textit{Network offered load} = \textit{flit\_size/flit arrival delay}$$

### A. Uniform Random Traffic Profile

The first set of simulations was performed with a 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 4 and Figure 5 show the average communication latency and throughput as a function of the average packet injection rate. As it can be observed from the results, QCA leads to a lower latency and higher throughput than the DyXY routing algorithm. This was expected due to the fact that our method considers local and global congestion information of the network in the routing decision. The algorithm can avoid the congested area, and thus reducing the average packet latency. The performance gain near the saturation point (0.8) is 28%.

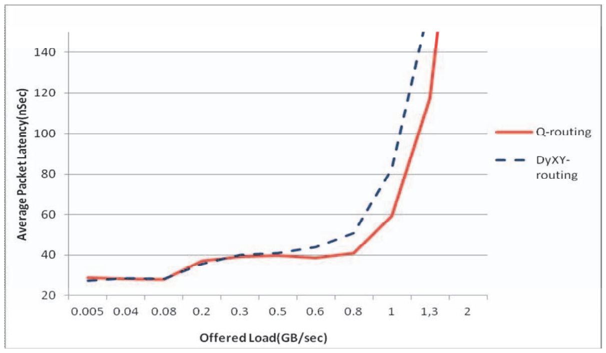

Figure 4. Average packet latency under uniform random traffic

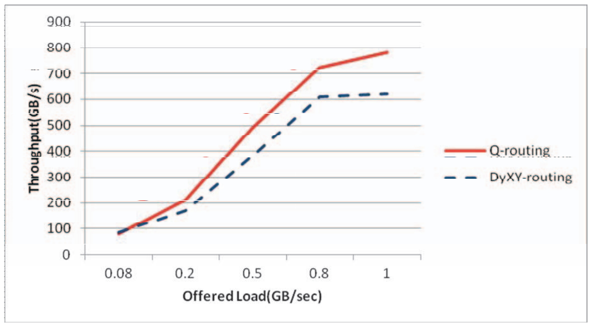

Figure 5. Throughput under uniform random traffic

## B. *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 the 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 node at *(1,2)* in the 4×4 2D-mesh network. The average packet latency and throughput of each network with *H=10%* are illustrated in Figure 6 and Figure 7, respectively. As observed from the results, QCA achieves better performance as compared to the DyXY routing algorithm. The performance gain near the saturation point (0.8) is 17%.

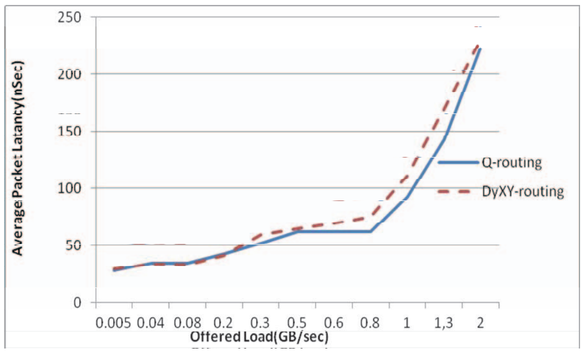

Figure 6. Average packet latency under hotspot traffic

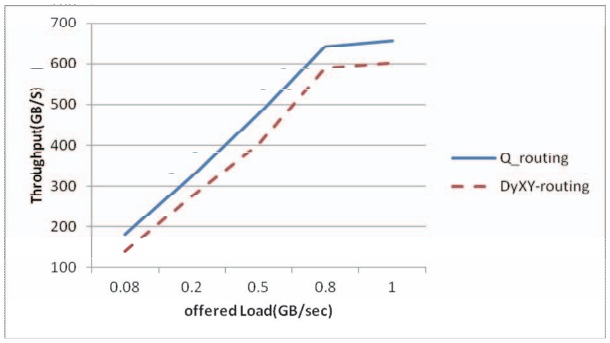

Figure 7. Throughput under hotspot traffic

## C. *Hardware Cost*

In this section, the hardware cost of our proposed method and DyXY schemes are computed together. 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 then again verified through post synthesis simulations. The layout areas of the two schemes are listed in Table III. As can be seen from the table, the area overhead of QCA is 12% higher that the DyXY method. That is because of using Q-table in each router to store local and global congestion information.

Table III. Hardware cost

| Components | Area ($mm^2$) |
|---|---|
| router in QCA method | 0.1683 |
| router in DyXY method | 0.1503 |

## VI. CONCLUSION

In this paper, we presented a congestion-aware routing algorithm (QCA: Q-learning based Congestion-aware Algorithm) for NoCs inspired by Q-learning techniques. The aim of the proposed method is to alleviate congestion of the network by a reliable estimation of the traffic status of the network. In this method, routers maintain distributed tables to store congestion information. These tables are updated by learning packets which carry local and nonlocal information between every two adjacent routers. The information in the tables contains updated values that can be used in the routing unit. Finally, in the routing decision unit, a less congested direction is selected for forwarding a packet. Experimental results with synthetic test cases demonstrate that the on-chip network utilizing the QCA method clearly outperforms a conventional on-chip network (Dynamic XY) considerably.

## VII. ACKNOWLEDGMENTS

The authors wish to acknowledge the Academy of Finland and Ulla Tuominen Foundation for the partial financial support during the course of this research.

## REFERENCES

[1] D. Wu, B. M. Al-Hashimi, and 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] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks", in Proc. of DAC, pp. 684-689, USA, 2001.

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

[4] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, et al., “A network on chip architecture and design methodology”, in Proc. of ISVLSI, pp. 117-124, USA, 2002.

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

[6] M. Kamali, L. Petre, K. Sere, and M. Daneshtalab, “Formal Modeling of Multicast Communication in 3D NoCs,” in Proceedings of 14th IEEE Euromicro Conference on Digital System Design (DSD), pp. 634-642, Sept 2011, Finland.

[7] M. Daneshtalab, M. Ebrahimi, P. Liljeberg, Juha Plosila, and H. Tenhunen, “A Low-Latency and Memory-Efficient On-hip Network,” in Proceedings of 4th IEEE/ACM International Symposium on Network-on-Chip (NOCS), pp. 99-106, May 2010, France.

[8] D. Kim, S. Yoo, S. Lee, “A Network Congestion-Aware Memory Controller”, in Proc. of Fourth ACM/IEEE International Symposium on Networks-on-Chip, pp. 257-264, 2010.

[9] P. Lotfi-Kamran, M. Daneshtalab, Z. Navabi, and C. Lucas, “BARP-A Dynamic Routing Protocol for Balanced Distribution of Traffic in NoCs to Avoid Congestion”, in Proc. of 11th ACM/IEEE Design, in Proc. of DATE, pp. 1408-1413, 2008.

[10] X. Dai, C. K. Li, and A. B. Rad, “An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control”, presented at IEEE Transactions on Intelligent Transportation Systems, pp. 285-293, 2005.

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

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

[13] 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.

[14] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, “GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks”, in Proc. of ISCA, pp. 194-205, 2003.

[15] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta, “Globally Adaptive Load-Balanced Routing on Tori”, IEEE Computer Architecture Letters, pp. 2-6, 2004.

[16] 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.

[17] J. Liang, S. Swaminathan, and R. Tessier, “aSOC: a scalable, single-chip communication architectures”, in IEEE Int. Conf. on PACT, pp. 37–46, Oct. 2000.

[18] J. Duato, S. Yalamanchili, and L. Ni, “Interconnection Networks: An Engineering Approach”, Morgan Kaufmann, 2002.

[19] 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.

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

[21] 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.

[22] 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, Hongkong.

[23] 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.

[24] L. T. Thiago, R. d. Rosa, F. Clermidy, et al., “Implementation and evaluation of a congestion aware routing algorithm for networks-on-chip”, pp. 91-96, 2010.

[25] 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.

[26] T.M. Mitchell,“Machine Learning”, McGraw-Hill Press, International Edition, 1997.

[27] S.Kumar, ”Condence based Dual Reinforcement Q-Routing: an On-line Adaptive Network Routing Algorithm”, pp. I98- 267, 1998.

[28] 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.

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

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