Microprocessors and Microsystems 38 (2014) 64–75

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Bi-LCQ: A low-weight clustering-based Q-learning approach for NoCs

F. Farahnakian *, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila

Department of Information Technology, University of Turku, Turku, Finland

ARTICLE INFO

Article history:
Available online 23 November 2013

Keywords:
Network-on-Chip
Congestion-aware routing algorithm
Adaptive routing algorithm
Q-learning and Q-routing approaches

ABSTRACT

Network congestion has a negative impact on the performance of on-chip networks due to the increased packet latency. Many congestion-aware routing algorithms have been developed to alleviate traffic congestion over the network. In this paper, we propose a congestion-aware routing algorithm based on the Q-learning approach for avoiding congested areas in the network. By using the learning method, local and global congestion information of the network is provided for each switch. This information can be dynamically updated, when a switch receives a packet. However, Q-learning approach suffers from high area overhead in NoCs due to the need for a large routing table in each switch. In order to reduce the area overhead, we also present a clustering approach that decreases the number of routing tables by the factor of 4. Results show that the proposed approach achieves a significant performance improvement over the traditional Q-learning, C-routing, DBAR and Dynamic XY algorithms.

1. Introduction

As the number of processing elements in a single chip increases, a new communication infrastructure is needed in place of the traditional bus-based architectures in a multiprocessor system-on-chip (MPSoC) design. Network-on-Chip (NoC) has been emerged as a dominant communication infrastructure in MPSoC design to meet better performance, flexibility, and scalability compared with previous solutions for on-chip communication [1–3]. To date, most NoCs have employed mesh topology because of its simple structure, ease of implementation, and support for reuse [4]. In a mesh-based NoCs, each core is connected to a switch by a local network interface. Each switch is connected to its neighbors through bidirectional links [5,6]. Cores can communicate with each other by propagating packets through switches in the network. For each packet, there might be several possible paths from any source to any destination. However, the underlying routing algorithm determines the selection between the possible paths and its required resources. This clearly demonstrates the impact of routing algorithms on performance. As the network size increases, congestion becomes one of the most important parameters limiting NoC performance. Adaptive routing algorithms partially address the network congestion problem by considering paths with low latencies [7]. These methods can alleviate congestion to improve the reliability and communication efficiency in NoCs. Adaptive routing algorithms, depending on the degree of adaptiveness can be classified as partially adaptive or fully adaptive. 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 to prevent deadlock [8]. Fully adaptive routing algorithms allow to route packets on any shortest path. To make the routing algorithm fully adaptive, there is a need of virtual channels in a wormhole switching network [9,10].

Reinforcement learning (RL) [11] is a machine learning paradigm that has been applied in many different areas. It obtains an optimal solution by trial-and-error interaction within a dynamic environment. In RL, a decision-maker or agent percepts the environment and chooses an action at each state. The agent receives a reward after every action it executes. The final goal of the agent is to learn a policy for selecting the best action among all possible actions. In other words, RL refers to a learning method that allows an agent to learn how to make a good decision from experiences and then improve the performance based on them. Q-learning is one of the most important breakthroughs in reinforcement learning while is developed by Watkins and Dayan [12]. In this free-model learning algorithm an 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. So Q-learning provides a self-optimizing controller design without a model of the environment. The Q-learning algorithm was used to create an adaptive routing algorithm named Q-routing [13]. In Q-routing, each switch makes its routing decision based on the information on the neighboring switches. A switch stores a table of Q-values that estimates the quality of alternative paths. These values are updated each time a switch sends a packet to one of its neighbors. This way, as

* Corresponding author.
E-mail addresses: fahfar@utu.fi (F. Farahnakian), masebr@utu.fi (M. Ebrahimi), masdan@utu.fi (M. Daneshtalab), pakrli@utu.fi (P. Liljeberg), juplos@utu.fi (J. Plosila).

0141-9331/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2013.11.008

the switch routes packets, Q-values gradually incorporate more global information.

In this paper, we propose a fully adaptive Q-routing based routing algorithm, named Bi-directional low-weight clustering-based Q-routing (Bi-LCQ). In this approach the network is split up into several clusters each maintaining a Q-table. This leads to not only reducing the area overhead considerably, but also providing a faster way of collection and utilization of local and global congestion information. The results show a significant performance improvement over traditional methods.

The reminder of this paper is organized as follows. Section 2 reviews the related work. Some background information on Q-learning and Q-routing techniques is given in Section 3. Section 4 describes the adjusted Q-routing algorithm for NoCs. In Section 5, the proposed adaptive routing algorithm is explained. The results are reported in Section 6 while the conclusion is given in the last section.

2. Related work

In recent years, a significant amount of research has been directed towards improving the routing efficiency of NoCs. The aim of many existing adaptive routing techniques is to distribute traffic over the whole network. 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 switches. The selection function selects an output channel from the set of channels supplied by the routing function [14]. The selection functions can be classified as either congestion-oblivious or congestion-aware schemes [15]. In congestion-oblivious algorithms, routing decisions are independent of the congestion condition of the network. For example, in the XY routing algorithm, packets first traverse along the X direction, then along the Y direction. In contrast, congestion-aware routing algorithms consider the congestion status of the network in their routing decisions. Several methods have been presented in order to address the network congestion problem [16,17]. 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 of other areas in the network. DyXY [18] uses local information, i.e. the current queue length of the corresponding input port in the neighboring switches, to decide on the next hop. DyXY may lead to forward packets through congested area as the utilized local information is not sufficient. Most common implementations of routing algorithms, e.g. NoP [19], RCA [15], and DBAR [20], have focused on collecting local or global congestion information to get an estimation of the congested areas in the network. An agent-based congestion-aware technique is presented in [21] to estimate congestion at some clusters and make the routing decision based on this estimation.

The Q-routing allows a network to be constantly adapted to the changing congestion condition and traffic flows. In this approach, each switch makes its routing decision based on the latency information, represented as a table of Q-values which estimate the quality of alternative routes. These values are updated each time a switch sends a packet to one of its neighbors. 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 and thus unreliable. Some proposals have been presented to address this issue such as CQ-routing [22], PQ-routing [23], and DRQ-routing [24]. 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. 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 is able to restore its previous policy. A similar idea is presented in DRQ-routing where the speed of restoration is expected to be high. In the DRQ method, learning is performed by carrying the latency information by data packets to intermediate switches (backward exploration unique to DRQ-routing) and also by receiving latency information from the neighboring switch where a data packet is sent to (forward exploration similar to Q-routing).

There are several works presented in NoCs using the Q-learning method. DyNoC is proposed in [25] to handle communication among modules which are dynamically placed on a reconfigurable NoCs. In another work, fault-tolerant deflection routing algorithm (FTDR) [26] is proposed inspired by Q-learning techniques for tolerating faults in NoCs. Q-routing is also used to provide different levels of Quality-of-Service (QoS) such as Best Effort (BE) and Guaranteed Throughput (GT) for NoCs [27]. 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. A Q-learning based Congestion-aware Algorithm (QCA) [28] is presented to alleviate congestion condition in NoCs. 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 (DuQAR) [29] 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 [30] 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. C-routing [31] is a Q-learning based method which takes advantage of clustering approach to reduce the routing table size compared with conventional Q-routing algorithms. Another work, CQ-routing [32], utilized also clustering method to improve the network performance and area reduction rather than C-routing.

In all of the aforementioned Q-routing methods, routing tables are updated by sending a data packet to a downstream switch and receiving an estimated latency value from the upstream switch. Therefore, as the distance between a source and destination increases, it takes more hops to update the routing table regarding each destination. In addition, Q-routing methods suffer from large area overhead in NoCs due to using large table sizes in each switch.

The goal of this work is to gain performance by applying the learning methods while keeping the area overhead as minimal as possible. In our proposed method, the network is divided into several regions, each region maintaining a routing table instead of each switch. This approach can reduce the table sizes by the factor of 4. It might be thought that this improvement is at the cost of degrading the performance. However, as shown in the result section, for instance, the performance is improved up to 45% over DyXY method in uniform traffic in 8×8 mesh network. The reason is that in traditional methods updating the routing tables are limited to the traversing path by the packet toward the destination. In Bi-LCQ, however, routing tables are updated by the congestion condition of the traversing regions. This provides a wider view of the hotspot areas. Moreover, in Bi-LCQ, routing tables are updated more frequently than traditional methods due to fast propagation of learning packets up to the factor of 3.

3. Background

In this section, we will give a brief introduction to the basic concepts of Q-learning as one of the most popular algorithms that

performs reinforcement learning. In addition, we describe the Q-routing algorithm taking advantage of the reinforcement learning aspect.

3.1. Q-learning

Q-learning [12] is a model-free learning method in which an agent can interact with the environment without prior knowledge about the system status to achieve a given goal. Thus, it is a highly adaptive and flexible algorithm. At each time, the agent observes the environment and chooses an action (a) based on the system state (s). By performing the action, the system moves from the current state (s) to the next state (s'). Then the agent receives a reward (a real number) or punishment (a negative reward) in the new state which updates the Q-value. Each Q-value is represented as $Q(s, a)$, which includes the expected long-term reward of taking action a in state s. Q-values can be stored in a table called Q-table. The Q-table maintains five fields named current state, next state, action, Q-value and goal state. *Current state* refers to everything that the agent currently perceives, while next *state* is determined by the current state and the actions selected by the agents. *Action* indicates the action that the agent performs. *Q-value* keeps the reward value that the agent receives in the current state when it executes an action. *Goal state* indicates the learning goal by trial-and-error.

The objective of Q-learning is to maximize the total future discounted rewards by mapping states to actions. There are two methods for selecting an action from the possible actions in every state [11] as below:

- *Exploration or random action selection:* Optimal actions, which are not chosen yet, are added to the table. Therefore, the agent chooses an action randomly.
- *Exploitation or Q-table based:* Actions are selected according to the learned Q-table.

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

3.2. Q-routing

Q-routing [13] is a network routing method based on the Q-learning method. It is able to find the least congested path among available paths from a source switch to a destination switch. 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 switch stores a Q-table that estimate the quality of alternative routes. Q-table is updated each time a switch sends a packet to one of its neighbors. Assume that the switch x sends a packet to one of its neighboring switches y, destined for the switch d (Fig. 1).

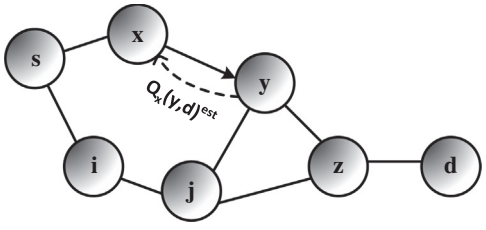

Fig. 1. An example of Q-routing method.

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: (1) the waiting time (qy) in the packet queue of the switch y (2) the transmission delay (δ) over the link from switch x to y, and (3) the time $Q_y(z, d)$ it would take for the switch y to send this packet to its destination via any of the switch y's neighbors (z). $Q_y(z, d)$ 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 of the y's neighboring switches.

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). Upon receiving the estimated value, the switch 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 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. The new Q-value in the Q-table is obtained by the following equation 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 Q-values. Learning rate (γ) determines at which rate the new information overrides the old one. Learning rate can take a value between zero and one; the value of zero means that no learning is made 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.*

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

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

3. *Forward the packet to neighboring switch y.*
4. *As 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.*

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

5. *Send y's estimate back to switch x which includes $Q_y(z, d)$ and qy.*
6. *Get ready for receiving the next packet (goto 1).*

- *Step 2*: when the switch x receives Q-values' estimate from its neighboring switch y.

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

4. Adjusted Q-routing algorithm for NoCs

In this section, we discuss about applying the traditional Q-routing algorithm to NoCs. At first, we adapt the Q-tables according to the mesh network features. Then we explain the format of data packet (propagating data information) and learning packet (for carrying the Q-routing information). Finally, we describe our proposed Q-routing algorithm for NoCs.

4.1. Adapted Q-table

Conventional Q-tables should be redefined in order to be used in NoCs. This modification is done as follows: *Current state* and *next state* represent the current and next switches in NoCs. *Action* refers to the output port which can be within one of the following cases: north, south, west, and east directions. *Q-value* is represented by the estimated congestion value to deliver a packet from the current to the destination switch via the next switch. Finally, *goal state* can be considered as the destination switch. In sum, *current state*, *next state*, *action*, *Q-value*, and *goal state* in conventional Q-tables are mapped into *current switch*, *next switch*, *output port*, *estimated latency*, and *destination switch* in NoCs, respectively.

Each switch in the network maintains a Q-table similar to Fig. 2(a). For example, this figure shows a Q-table of switch 1 in 8×8 mesh 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; and the next switch is the neighboring switch having the minimum latency value for a corresponding destination. Since a separate row should be dedicated to each destination in the network, the area overhead of Q-tables becomes problematic in NoCs.

Notice that, in conventional Q-routing models, the waiting time at input buffers is a measure of latency, however, we use the number of occupied buffer slots at input buffers as a metric of latency. C-routing algorithm [31] is a Q-routing based approach proposed in NoCs. It defines a new Q-table structure (CRouting-tables) in order to reduce the Q-table sizes. In the C-routing approach, the number of columns is $m-1$, where m is the number of dimensions. By using a clustering approach (it will be explained in details in Section 5), the number of rows can be decreased to $(n/C)+C$, where n is the number of switches in the network and C is the number of clusters. In sum, each switch in the C-routing algorithm maintains a CRouting-table (Fig. 2(b)) with the size of $((n/C)+C)\times(m-1)$. As an example, in 8×8 mesh network, the table size is reduced by 75% compared with conventional Q-tables. Our basic structure of the adapted Q-table (AQ-table shown in (Fig. 2(c)) reduces the number of columns to two. The reason is that in a minimal routing, packets can be delivered into at most two directions at a switch. Therefore, there is no need to maintain four columns as two columns are always empty.

4.2. Data and learning packets formats

Two types of packets traverse in the network, data packets and learning packets. They use separate virtual channels to propagate information. The header flit format of a typical data packet is shown in Fig. 3. Each flit is n-bit wide including 6-bit source address, 6-bit destination address, 1-bit EOM (End of Packet), and 1-bit BOM (Begin of Packet) sign. The content of the packet is located in the rest of the flits (body).

The learning packet is a 1-flit packet that is generated for per data packet. As shown in Fig. 4, it is a 1-flit packet consisted of four main fields described as follows:

- *Direction:* It is a 1-bit value determining the direction from which a learning packet is forwarded. The value of 0 and 1 are corresponding to the X-direction and Y-direction, respectively.
- *NewEstimatedCongestion:* It is obtained by adding the local and global latencies. Local latency determines the occupied buffer slots in all virtual channels of the input buffer. In Eq. (1) the term q_y is referred to local latency. Global latency determines the estimated latency from the next switch to the destination switch. In Eq. (1), the term $Q_y(z, d)$ indicates the global latency. We considered 4 bits to encode the total estimated latency.
- *Destination Switch ID:* It determines the destination switch of a data packet. 6 bits are considered for this purpose which is

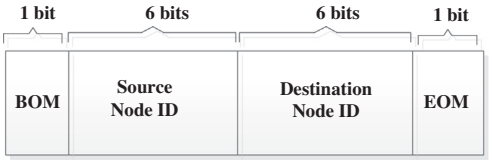

Fig. 3. Header of the data packet format.

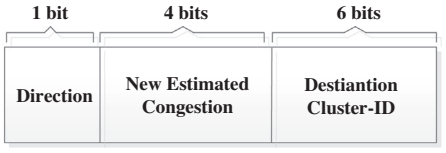

Fig. 4. Learning packet format.

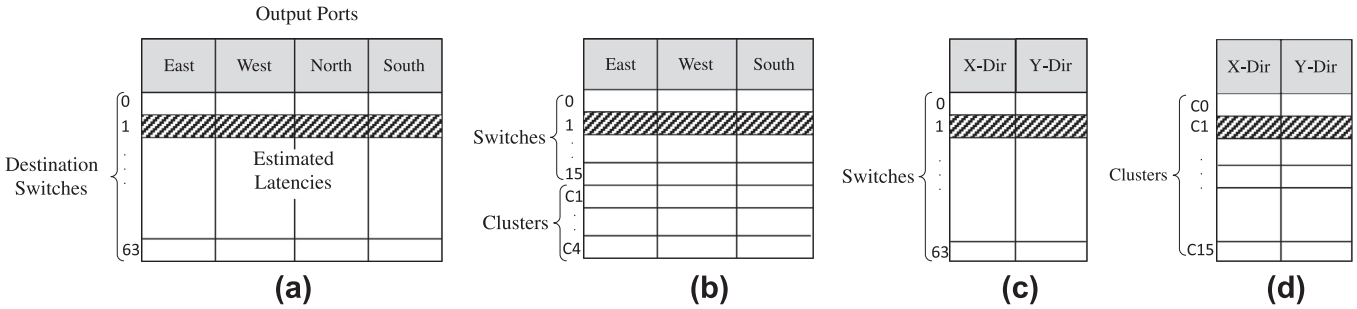

Fig. 2. Formats of different Q-tables in 8×8 mesh network (a) Q-table at switch 1, (b) CRotuing table at switch 1, (c) AQ-table at switch 1, and (d) CQ-table at cluster 1.

sufficient for 8×8 mesh network. For a bigger size network, more bits should be allocated to the destination ID field.

4.3. Q-routing policy

Routing units decide from which output channel a packet should be delivered. The decision is made based on the latency values stored in Q-tables. So, it is required to select between Q-values and also keep Q-tables updated. *Minimum selection function* and *update Q-table function* are responsible for these tasks. By receiving a data packet, the *minimum selection function* selects an output channel with the lowest latency value toward the destination. Upon receiving the learning packet, the *update Q-table function* updates the corresponding entry in Q-table with a new value. In the following pseudo code, the usages of both functions are illustrated when a data packet is delivered from switch x to switch y and a learning packet is transferred from switch y to switch x.

1. *At switch y, the **minimum selection function** returns one of the neighbors of switch y with the smallest Q-value.*
2. *The packet is sent from the current switch y to the downstream switch; concurrently a learning packet is delivered to switch x.*
3. *Switch x receives the learning packet and updates Q-value in the corresponding row of Q-table, calling the **update Q-table function**.*

In minimum selection function, when the switch x receives a packet destined for the switch d, it checks the vector $Q_x(*; d)$ of Q-values to select a neighboring switch y in which $Q_x(y; d)$ is minimum (note that $*$ determines all neighboring switches of switch 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 switch and thus the routing decision based on these estimates are close to optimal solution but does not necessarily give the best solution.

```
1.  Dest_id ← Get_Destination_Switch();
2.  if destination is the local switch then
3.      consume the packet
4.  end if;
5.  if (number of neighboring switches=1) then
6.      the packet is sent through the only possible direction
7.  end if;
8.  if (number of neighboring switches=2) then
9.      if (Q-table(Dest_id)(X-dir) < Q-table(Dest_id) (Y-dir))
10.         SelectedNeighbor ← NeighboringSwitch _Xdir;
11.     else
12.         SelectedNeighbor ← NeighboringSwitch_Ydir;
13.     end if;
14. end if;
```

When a learning packet arrives to a switch (e.g. switch x), Q-table is updated in order to be adapted to the changing state of the network.

```
By receiving the learning packet, extract the header
1. Dir ← get_direction()
2. Q_value_new ← get_NewEstimated_latency()
3. Dest_id ← get_Destination_Switch_id()
        ------------------
4. Q_value_old ← Q_table(dest_id)(Dir)
5. Q-table(dest_id)(Dir) ← Q_value_old + 0.5(Q_value_new -
   Q_value_old)
```

As already discussed, Eq. (1) is used to calculate the new estimated latency. Without loss of generality, in this work, we assume that the link delay, δ, is a constant value of zero in Eq. (1). Moreover, a simple weighting assignment of 50–50 is used for old and new information. Therefore, Eq. (1) is rewritten as:

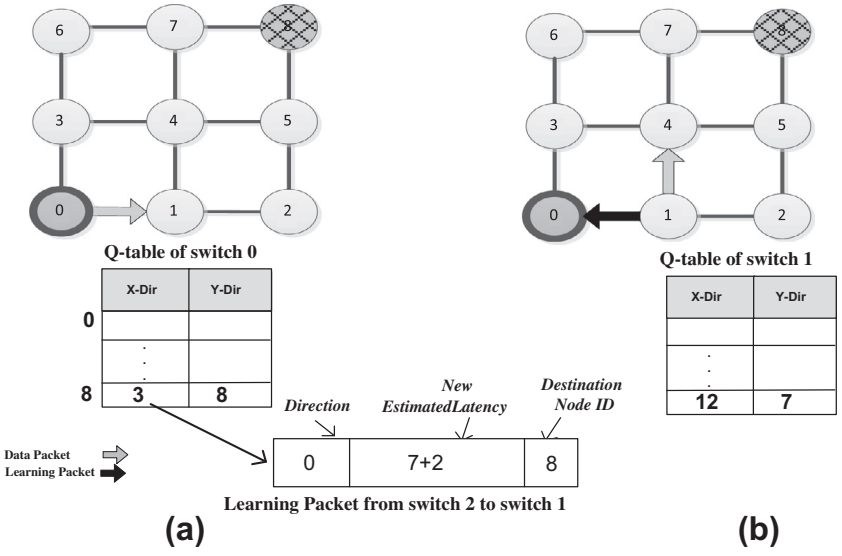

Fig. 5. An example of adjusted Q-routing for NoCs.

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

Fig. 5(a) illustrates an example where a data packet is going to be sent from switch 0 to the destination switch 8. The packet can be delivered through either switch 1 or switch 3 which are located on X and Y directions, respectively. As shown in the Q-table of switch 0, the latency value in the X-direction is smaller than the Y-direction, so the packet is forwarded to switch 1.

Whenever, the data arrives at the intermediate switch 1 (Fig. 5(b)), the routing unit decides whether to send a packet to switch 2 or switch 4. Suppose that the routing unit selects the neighboring switch in the Y-direction (i.e. switch 4) as the next hop which has a lower latency value. At this time, a learning packet is generated and delivered back to switch 0. This learning packet should aggregate the local and global congestion information before sending to switch 4. To obtain the local congestion information, the number of occupied buffer slots at the input buffer of switch 1 is used. For the global information, the latency value is extracted from the corresponding row of the Q-table at switch 1. This value is the minimum estimated latency required to send a packet from switch 1 to the destination switch 8 through switch 4 (i.e. along the Y-direction).

In our example, the local and global congestion information is assumed to be 2 and 7, respectively. After summing up the local and global congestion information, the obtained value is delivered back to switch 0 via a learning packet. Upon receiving the learning packet, switch 0 updates its Q-table by using the old and new estimated values. In this example, this modification affects the row number 8 and the X-Dir column at the Q-table of switch 0.

Although Q-routing is able to alleviate congestion by choosing a low congested path among alternative paths, it suffers from high area overhead. Each switch requires a Q-table with the size of $(n^2 \times 4)$ in a $n \times n$ mesh network. Therefore, in the whole network, the area overhead of using Q-tables is equal to $n^2 \times (n^2 \times 4)$. This unacceptable area overhead motivates us to present a low-weight Q-routing approach. This method is not only able to reduce the area overhead significantly, but also improve the performance.

5. Low-weight clustering-based Q-routing

As already mentioned, the original Q-routing is not suitable for NoCs, as it needs a large area for storing Q-tables. In this section, to tackle with the problem of area overhead, we present a novel low-weight clustering-based Q-routing approach, called LCQ. The Q-tables in this approach are called Cluster Q-tables (CQ-tables). In the proposed clustering approach, the size of the traditional Q-table is decreased such that it can be employed in NoCs.

An idea is to consider the network as different regions (clusters) each containing a single CQ-table for all the switches within the cluster instead of maintaining a CQ-table for each switch. CQ-tables maintain the information about the routing cost of sending a packet from the source cluster to the possible destination clusters. Therefore, the CQ-table contains $C-1$ rows and 2 columns as shown in Fig. 2(d), where C is the number of clusters in the network. The Q-values in each row indicates the congestion values from the current cluster to reach the destination cluster through each of the neighboring clusters in X or Y direction.

5.1. Data and learning packets' format

In LCQ, the data packet format is modified by adding 6 bits into the header flit (Fig. 6). These additional fields can be described as follows:

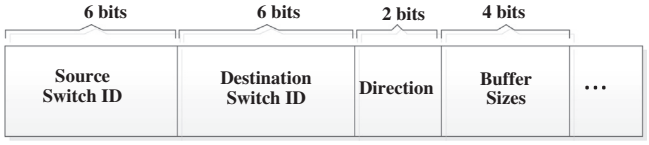

Fig. 6. Header of the data packet format.

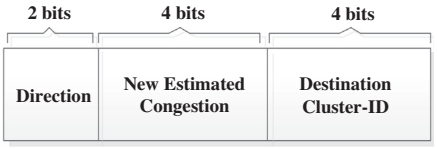

Fig. 7. Learning packet format.

- *Direction:* determines the direction in which the data packet is forwarded from the sender cluster into the receiver cluster. Since each cluster is connected to at most four neighboring clusters, two bits are enough to encode the direction of neighboring cluster IDs.
- *BufferSizes:* determines the average number of occupied buffer slots of the input buffers inside a cluster from where the packet enters the cluster until it leaves the cluster. We have assigned four bits for this field.

The learning packet consists of three fields as shown in Fig. 7. These fields are described as follows:

- *Direction:* determines the direction of receiver cluster ID of the data packet or direction of sender cluster ID of the learning packet. Two bits are enough for this field.
- *NewEstimatedCongestion:* It is obtained by summing up the local and global congestion values. Local congestion is measured by dividing the sum of the occupied buffer slots into the number of hops taken by a packet within a cluster. The congestion value is determined with four bits.
- *DestinationCluster-ID:* determines the destination cluster ID. 4 bits are considered for this field.

The receiver cluster of the learning packet uses the *NewEstimatedCongestion* value to update the CQ-table. The *DestinationCluster-ID* determines the row and by using *Direction* the column is determined.

5.2. Routing algorithm

In this section, we describe the functionality of the routing algorithm, and how a packet is routed from a source switch to a destination switch. A clustered network in an 8×8 mesh network is illustrated in Fig. 8 in which the network is divided into 16 clusters. The number of switches within each cluster is considered to be four as the traffic condition within each cluster is roughly similar.

Clusters can be also formed dynamically at run time, so that based on the traffic condition, the numbers of switches within each cluster changes dynamically. This claims that all Q-tables be reset and reformed according to the new cluster's configuration even if only one switch leaves its cluster and gets involved to another one. Thereby, another learning phase is needed to fill out the Q-tables. Moreover, the number of rows in Q-tables should be set with the maximum number of clusters in the network. There are many other challenges behind dynamic clustering such as: defining the maximum number of switches that can be included in a cluster

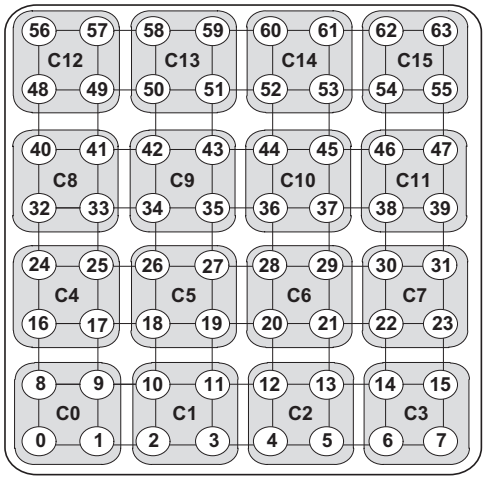

Fig. 8. Clustering approach in an 8 × 8 mesh.

and the metrics for dividing switches into different clusters. We consider the dynamic clustering approach as our future work.

In LCQ, every switch first determines the destination switch of the received packet. If the destination switch is located in the same cluster as the current switch, it uses the XY routing algorithm because the destination switch is close to the current switch and it does not need a more complex routing algorithm to cope with. Otherwise, if the destination switch is located in another cluster than the current switch, LCQ chooses a neighboring cluster which has a lowest latency value.

Suppose that a packet is generated at C0 toward the destination C15. This packet has just passed through cluster C5 and it is already at cluster C6. The packet can be either sent through cluster C7 or C10. The decision is made based on the latency values stored in the CQ-table of cluster C6. Let us assume that cluster C7 has lower latency value. The packet is needed to passing though cluster C6 to reach cluster C7. By routing a packet inside cluster C6, the number of occupied buffer slots of the input buffers (that are located in the path) is stored in the header flit and carried by the packet. Upon forwarding the packet to the neighboring cluster C7, a learning packet is generated at cluster C6 and delivered back to the cluster C5. The learning packet includes the local and global latencies which can be seen as a new estimated latency of sending a packet from cluster C5 to the destination switch via cluster C6. By receiving the learning packet at cluster C5, the old latency estimation is combined with the new value and the CQ-table is updated. Therefore, Eq. (1) would be:

$$Q_{C5}(C6,Cd)_{new} = Q_{C5}(C6,Cd)_{old} + 0.5(Q_{C6}(C7,Cd) + q_{C6} - Q_{C5}(C6,Cd)_{old}) \quad (2)$$

The difference between Eqs. (1) and (2) is that, Eq. (1) is defined at the switch level while Eq. (2) is defined at the cluster level. For example, the local latency in Eq. (1) determines the occupied buffer slots at the input buffer of a neighboring switch, while in Eq. (2) it is a sum of occupied buffer slots of the input buffers along the path in the neighboring cluster (q_{C6}). In Eq. (2), the global latency ($Q_{C6}(C7, Cd)$) indicates the minimum estimated latency to reach from cluster C6 to the destination switch via one of the neighboring clusters (C7). In general, each cluster is aware of the traffic load in neighboring clusters and selects the least congested cluster for avoiding overloaded links. Similar to Q-routing, the proposed algorithm can be summarized in two steps: when a switch receives a data packet and when it receives a learning packet.

Step 1: the functionality of LCQ algorithm is described in the following pseudo code when a switch has just received a packet in current cluster *Cc*. This packet is going to be sent to a switch in the destination cluster *Cd*. After receiving a data packet by a switch in cluster *Cc*, the switch compares the current and destination cluster IDs. If cluster IDs are the same, XY routing is employed, otherwise learning method is utilized (lines 1–7).

```
1.  Determine the current cluster(Cc);
2.  Determine the destination cluster(Cd);
3.  if Cc = Cd then
4.     Use xy_routing_algorithm;
5.  else
6.     Use Q_learning_routing_algorithm;
7.  end if;
    ------------------------------------------------
8.  function Q_learning_routing_algorithm
9.  –Extracting information from data packet header
10.       Cu ← UpstreamCluster_ID;
11.       BufferSizes ← BufferSizes;
12. –Determining the neighboring cluster (Cn) with minimum
    traffic load
13.          Cn = min_{Cm ∈ Neigbors(Cc)} Q(Cm, Cd)
14. –Measuring total occupied buffer slots in input buffers
    when the packet traveling inside cluster Cc
15.    Do
16.    BufferSizes ← OccupiedBufferSlotsinInputBuffer +
    BufferSizes;
17.    Forward packet to the neighboring switch;
18.    Until (packet is going to leave the cluster Cc);
19. –Upon leaving the packet, generate learning packet and
    send it back to upstream cluster Cu
20.    LocalLatency ← BufferSizes /NumberOfHops;
21.    GlobalLatency ← Q_Cc(Cn,Cd);
22.    NewEstimatedLatency ← LocalLatency + GlobalLatency;
23.    RecivingCluster_ID ← Cu;
24. DestinationCluster_ID ←Cd;
25. –Updating the data packet header before delivering packet
    into cluster Cn
26.    UpstreamCluster_ID←Cc;
27. BufferSizes ← 0;
28. End function
```

If the routing algorithm is Q-learning, *UpstreamCluster_ID* and *BufferSizes* fields are extracted from the data packet header (lines 8–11). Meanwhile, a neighboring cluster with a minimum estimated latency is selected from the CQ-table (lines 12–13). By traversing the packet within a same cluster, the occupied buffer slots (BufferSizes) in the input buffers are summed together and carried by the header flit (lines 14–18). Upon leaving the current cluster, the learning packet is generated and delivered to the upstream cluster *Cu*. The learning packet should carry the new estimated latency which consists of the local and global latencies. The local latency is obtained by dividing the sum of occupied buffer slots to the number of hops taken by a packet inside cluster *Cc*. The global latency is extracted from the CQ-table of the current cluster *Cc*. This value shows the estimated latency of the packet in the remaining path from the next cluster *Cn* to the destination cluster *Cd*. These values are used to update the corresponding entry of the CQ-table in the upstream cluster *Cu*. The row of the table is determined by a destination cluster ID while the column is a direction in which the data packet has been already delivered from (lines 19–24). Before sending the data packet to the next cluster, some modifications should be made in the header flit such as setting the upstream cluster and resetting the occupied buffer slots value (lines 25–28).

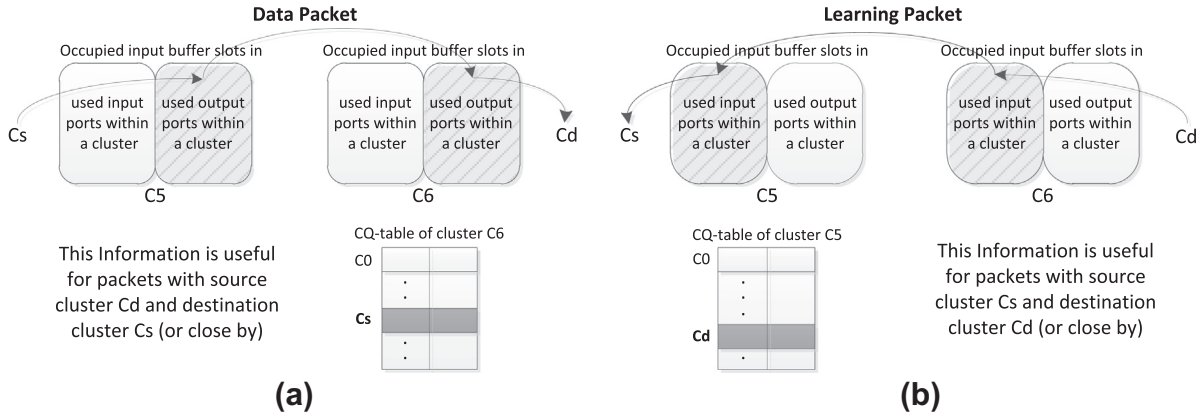

Fig. 9. An example of Bi-LCQ.

Step 2: The functionality of LCQ algorithm is described in the following pseudo code when a switch in the cluster receives the learning packet. As shown in this algorithm, the switch extracts the new estimated latency, column and row information from the learning packet (lines 1–9) and then updates the CQ-table using the old and new estimation values (lines 10–13). We use the round-robin arbiter to allow access to a CQ-table when there are multiple requests for updating a CQ-table in the related cluster.

```
1.  –Extracting information from learning packet
2.     column ← direction(ReceivingCluster_ID);
3.     row ← DestinationCluster_ID;
4.  if (xReceivingCluster-xCurrentCluster != 0)
5.     column ← xDir;
6.  else
7.     column ← yDir;
8.  end if;
9.     Q_est ← NewEstimatedLatency;
10. –Update the corresponding entry of the CQ-table using Eq. (2) as follow:
11.    Q_old ← CQ_table(row,column);
12.    Q_new ← Q_old + 0.5 (Q_est + Q_old);
13. CQ_table(row,column) ← Q_new;
```

In LCQ, each cluster makes its routing decisions based on Q-values that estimate the quality of alternative routes. These values are updated each time the cluster sends a packet to one of its neighboring clusters. If a Q-value is not updated for a long time, it does not reflect the real congestion status of the network and the routing decision based on unreliable Q-values cannot be accurate. To address this issue, we use a new method for updating Q-values more frequently. We called it Bi-directional LCQ (Bi-LCQ) because each cluster updates Q-values by the means of not only learning packets but also data packets. In other words, similar to a learning packet, a data packet collects the congestion information along its path within a cluster. Then, this information is used to update a Q-value in the next cluster. In Bi-LCQ, data packets are also employed to update Q-values as they move from the source to destination cluster. Unlike learning packets that collect the number of occupied buffer slots from the input buffer of input port, data packets collect the information from the input buffer of output port. Fig. 9(a) shows an example where a data packet is sent from cluster *C*5 to cluster *C*6 while in Fig. 9(b), a learning packet is delivered from the cluster *C*6 to cluster *C*5. The source and destination clusters are *Cs* and *Cd*, respectively. Due to simplicity, the boxes are represented as the total number of occupied buffer slots of the input buffers on input ports and output ports within a cluster. As shown in Fig. 9(a), the data packet collects the congestion values on the input buffers related to the output ports. The reason is that this information is useful in future for the packets traversing in the reverse path (i.e. from cluster *Cd* to cluster *Cs* or close by). By using this information in cluster *C*6, the row *Cs* of the CQ-table is updated. When the data packet arrives to cluster *C*6, a learning packet is delivered back to cluster *C*5. The learning packet, however, carries the congestion information of the input buffers of the input ports. This information is used to update the row *Cd* of the CQ-table in cluster *C*5.

6. Result and discussions

To evaluate the efficiency of the proposed routing scheme, we selected three algorithms DBAR [20], DyXY [18] and C-routing [31] schemes which are explained in the related work. A wormhole-based NoC simulator based on the OMNET framework [33] is used to model all major components of the on-chip network and simulations are carried out to determine the latency characteristics of each network [34]. The simulator inputs include the network size, packet size, the number of virtual channels, the network offered load, and the traffic type. In each simulation time step (nanosecond), packets are stored in the FIFO buffer of the source switches. The amount of injected packets depends on the network offered load. Below is the equation for calculating the network offered load:

$$Networkofferedload = flit\_size/flit\text{-}arrival\text{-}delay$$

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 previous flit generation and next flit generation.

The simulations were conducted on 8×8 and 14×14 meshes under various traffic patterns. For each simulation, only data packet latencies are averaged over 16,000 packets. Since learning packets exists only for one hop between the neighboring switches, their latencies are very small compared with data packets traversing many hops. For this reason, in the results, latencies are calculated for data packets and averaged over the number of them. To propagate data packets, two virtual channels are used along both dimensions, while a separate virtual channel is allocated to learning packets. Three synthetic traffic profiles including uniform random, one-switch and multi-switch hotspot, along with SPLASH-2 [35] application traces are used.

6.1. Learning process

Learning process can be divided into two phases:

- *Training phase*: training phase is an essential important step to model a learning system in an unknown environment. The learning system obtains a good knowledge about the given task in this phase. This knowledge is utilized in testing phase which results in improved performance.

Since the proposed learning method makes decisions based on the Q-values, the values should be kept up-to-date. We plotted the variation of Q-values under the hotspot traffic with a single hotspot at switch (4, 4) and (7, 7) in the 8×8 and 14×14 mesh. The variation of Q-values is plotted on the vertical axis and simulation time on the horizontal axis. As observed from Figs. 10 and 11, Q-values change abruptly at the beginning of the training phase. In Fig. 10 after 30,000 ns, the variation of Q-values is almost stable and the network is fully trained. Therefore, 30,000 ns can be considered as the training time in an 8×8 mesh. As shown in Fig. 11, the variation of Q-values reaches to a stable phase after 45,000 ns in 14×14 mesh. It means that more training time is needed as the network size enlarges, about 3000, 4500 packets are generated in the 8×8 and 14×14 mesh, respectively. In sum, the simulations are "warmed up" by 3000 and 4500 packets, which is enough to reach the reliable Q-values.

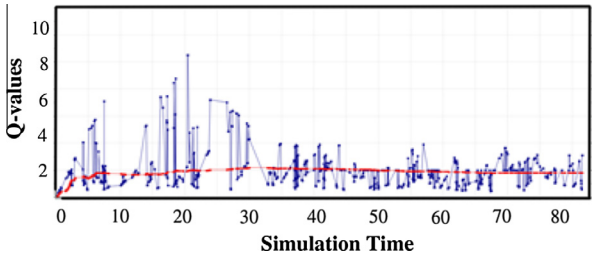

Fig. 10. The variation of Q-values a hotspot switch at (4, 4) in the 8×8 mesh.

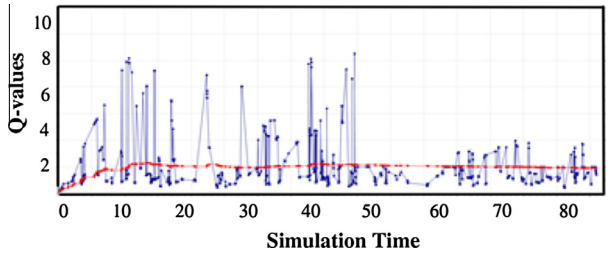

Fig. 11. The variation of Q-values of a hotspot switch at (7, 7) in the 14×14 mesh.

- *Testing phase*: In this phase, the decision is made based on the knowledge that is collected in training phase. Learning is also made in this phase as the network condition changes dynamically. This phase continues until the end of simulation time. In general, setting the sufficient time for these phases depends on the problem and it cannot be easily predicted.

6.2. Uniform random traffic profile

In the uniform traffic profile, a switch sends a packet to other switches with a uniform distribution. In Fig. 12, the average latency as a function of the offered load is plotted for both mesh sizes. As observed from the results, the Q-routing schemes (LCQ, Bi-LCQ and C-routing) behave as efficiently as DBAR and DyXY especially in medium and high loads. As load increases, DBAR is unable to tolerate the high load condition, while the Q-routing schemes learn an efficient routing policy. Bi-LCQ leads to the lowest latency due to the fact that it can distribute traffic more efficiently than the other Q-routing schemes. In fact, in DBAR, C-routing (CR) and Q-routing (QR), packets use minimal paths based on only local congestion information in adjacent switches. But in LCQ and Bi-LCQ methods, routing decision depends on congestion information in adjacent area (clusters).

6.3. Hotspot traffic profile

Under the hotspot traffic profile, 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 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 meshes, respectively. The average latency of each network with $H = 20\%$ are illustrated in Fig. 13.

As observed from the figure, the proposed routing scheme achieves better performance compared with the other schemes. The Bi-LCQ method can alleviates the congested areas and performs considerably better than other schemes. Using minimal routes along with the intelligent routing policy reduces the average network latency of Bi-LCQ.

Also, we evaluate the multi-switch hotspot in the center of network at (4, 4), (3, 4), (3, 3) and (4, 3) for 8×8 mesh and at

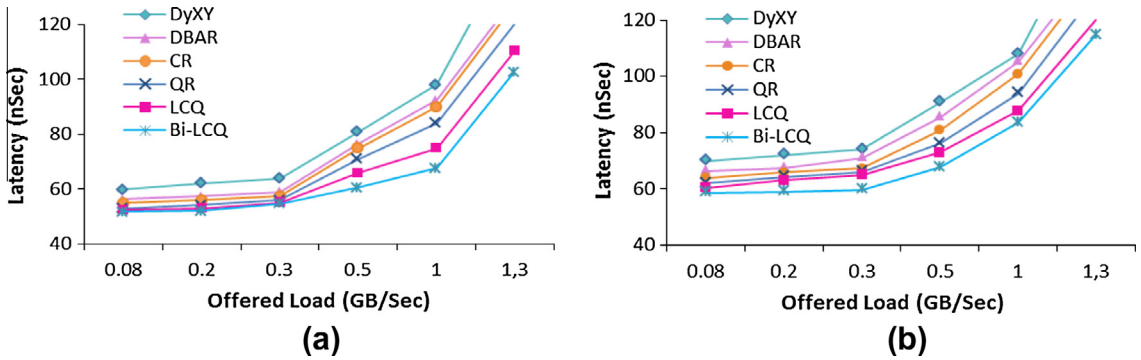

Fig. 12. Performance under uniform traffic model in (a) 8×8 mesh and (b) 14×14 mesh.

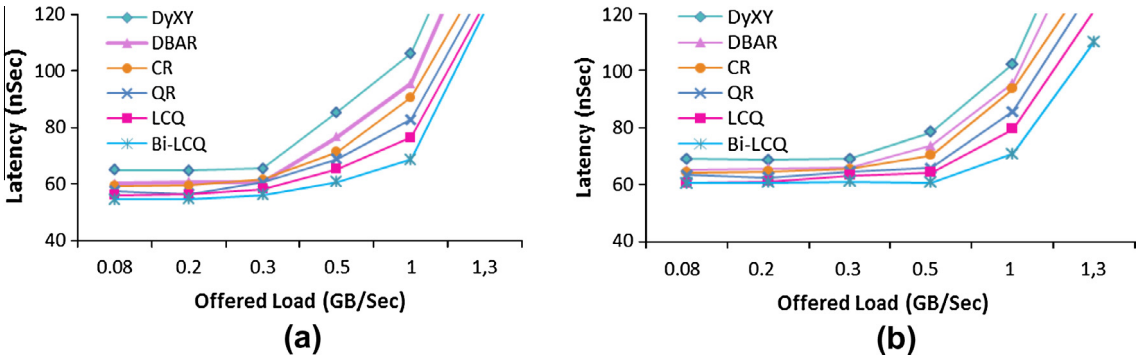

Fig. 13. Performance under one-switch hotspot traffic model in (a) 8 × 8 mesh and (b) 14 × 14 mesh.

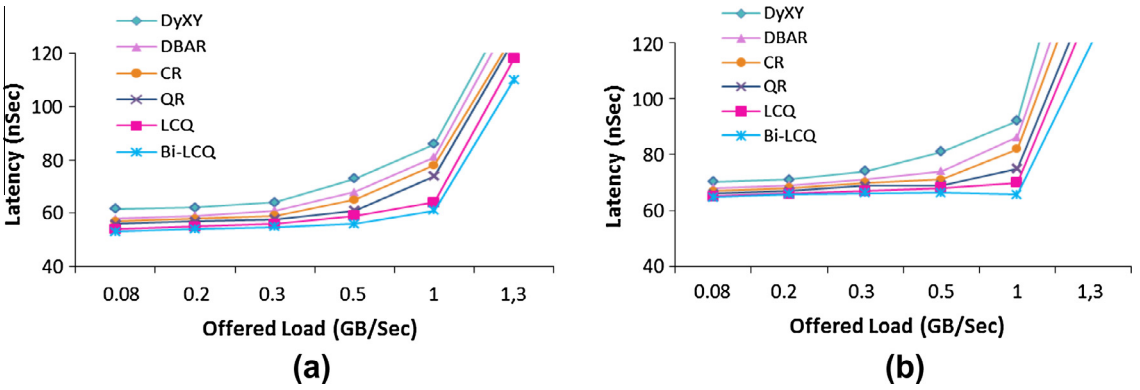

Fig. 14. Performance under multi-switch hotspot traffic model in (a) 8 × 8 mesh and (b) 14 × 14 mesh.

(7, 7), (6, 7), (6, 6) and (7, 6) for 14 × 14 mesh. Fig. 14 shows that Bi-LCQ achieves better performance compared with the DyXY routing, DBAR and the other Q-routing techniques at high traffic load. This is because the Q-values are frequently updated and thus reflecting the real status of the network when the network gets congested.

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

In addition, the impact of using Q-Learning policy and clustering approach near the saturation point (0.5) over Q-routing, C-routing, DBAR and Dynamic XY-routing algorithms is summarized in Table 2 for an 14 × 14 mesh. The results reveal that usage the proposed approach distributes the traffic efficiently.

6.4. Application traffic profile

Application traces are obtained from the GEMS simulator [36] using some application benchmark suites selected from SPLASH-2. We use a 64-node 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-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 64B. 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.

Table 1
Performance gain of the Bi-LCQ method over others methods for three traffic patterns in an 8 × 8 mesh.

Traffic pattern	Low-weight cluster-based Q-routing (LCQ) (%)	Q-routing (QR) (%)	C-routing (CR) (%)	DBAR (%)	DyXY (%)
Uniform	14	23	27	32	45
Hotspot (one switch)	11	19	22	27	38
Hotspot (multi-switch)	9	17	19	24	36

Table 2
Performance gain of the Bi-LCQ method over others methods for three traffic patterns in an 14 × 14 mesh.

Traffic pattern	Low-weight cluster-based Q-routing (LCQ) (%)	Q-routing (QR) (%)	C-routing (CR) (%)	DBAR (%)	DyXY (%)
Uniform	10	12	19	26	34
Hotspot (one switch)	9	10	16	21	30
Hotspot (multi-switch)	7	13	18	24	28

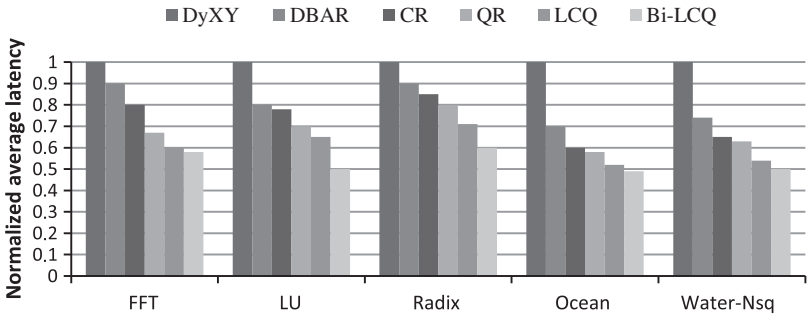

Fig. 15. Performance across SPLASH-2 benchmarks.

Fig. 15 shows the average packet latency across five benchmark traces, normalized to DyXY. Bi-LCQ provides lower latency than other schemes and it shows the greatest performance gain on Water-Nsq with 49% reduction in latency.

6.5. Area overhead

As already discussed, the size of the Q-table can be reduced by using the C-routing table. The size of the CQ-table can be significantly decreased over Q-table and C-routing tables by the LCQ approach. The occupied area by the Q-tables in $n \times n$ mesh network for Q-routing, C-routing and LCQ can be calculated with the following equations:

$$T_{Q\text{-}routing} = n \times (n \times m)$$

$$T_{C\text{-}routing} = n \times ((D + C) \times (m - 1))$$

$$T_{CQ\text{-}routing} = \frac{n}{D} \times ((C - 1) \times 2))$$

In these equations, m is the number of directions, C is the number of clusters, and D indicates the number of switches within each cluster. The required area of each method in 8×8 mesh is given in Table 3. As can be observed from the table; the size of Q-tables has been considerably reduced by our approach.

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 4. The area overhead of Bi-LCQ is smaller than Q-routing and C-routing methods.

Table 3
The area overhead in an 8×8 mesh.

Routing method	No. of clusters	No. of tables	Occupied area in network
Q-routing	0	64	$64 \times 64 \times 4 = 16{,}384$bits
C-routing	4	64	$64 \times (16 + 4) \times 3 = 3840$bits
LCQ	16	16	$16 \times (15 \times 2) = 480$bits

Table 4
Hardware cost.

Routing method	Network area (mm^2)
Q-routing	3.913
C-routing	3.187
LCQ	3.025
Bi-LCQ	3.095

7. Conclusion

In this paper, we proposed a novel congestion-aware routing algorithm based on Q-learning. The presented routing algorithm split up the network into several clusters each maintaining a CQ-table. This table stores local and global congestion information about alternative routes for forwarding a packet to the destination cluster. Each cluster can select the less congested output channel based on CQ-table information. Moreover, in order to update CQ-tables more frequently, both learning and data packets take part in propagating the congestion information. By our proposed approach, not only the area overhead but also latency is reduced significantly. The results show a significant performance improvement over traditional methods.

References

[1] R. Marculescu, U.Y. Ogras, Li-Shiuan Peh, N.E. Jerger, Y. Hoskote, Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives, IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 28 (2009) 3–21.
[2] S. Murali, M. Coenen, A. Radulescu, K. Goossens, G. De Micheli, A methodology for mapping multiple use-cases onto networks on chips, Proc. Des. Autom. Test Eur. Conf. 2006 (2006) 118–123.
[3] H.G. Lee, N. Chang, U.Y. Ogras, R. Marculescu, On-chip communication architecture exploration: a quantitative evaluation of point-to-point, bus, and network-on-chip approaches, ACM Trans. Des. Autom. Electron. Syst. 12 (2007) 20–40.
[4] J. Liang, S. Swaminathan, R. Tessier, aSOC: a scalable, single-chip communication architectures, in: IEEE Int. Conf. on PACT, October 2000, pp. 37–46.
[5] W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: Proc. of DAC, USA, 2001, pp. 684–689.
[6] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, IEEE Comput. (2002) 70–78.
[7] D. Kim, S. Yoo, S. Lee, A network congestion-aware memory controller, in: Proc. of Fourth ACM/IEEE International Symposium on Networks-on-Chip, 2010, pp. 257–264.
[8] G.-M. Chiu, The odd-even turn model for adaptive routing, IEEE Trans. Parallel Distrib. Syst. 1 (7) (2000).
[9] Y.M. Boura, C.R. Das, Efficient fully adaptive wormhole routing in n-dimensional meshes, in: Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS), 1994.
[10] A.A. Chien, J.H. Kim, Planar-adaptive routing: low-cost adaptive networks for multiprocessors, J. ACM 42 (1) (1995).
[11] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 2000.
[12] C.J.C.H. Watkins, P. Dayan, Q-learning, in: Proc. Machine Learning, 1992, pp. 279–292.
[13] J.A. Boyan, M.L. Littman, Packet routing in dynamically changing networks: a reinforcement learning approach, Adv. Neural Inform. Process. Syst. 6 (1994) 671–678.
[14] J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks: An Engineering Approach, Morgan Kaufmann, 2002.
[15] P. Gratz, et al., Regional congestion awareness for load balance in networks-on-chip, in: Proc. HPCA, 2008, pp. 203–214.
[16] A. Singh, W.J. Dally, B. Towles, A.K. Gupta, Globally adaptive load-balanced routing on tori, IEEE Comput. Archit. Lett. (2004) 2–6.
[17] 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, 2005, pp. 559–564.
[18] M. Li, Q. Zeng, W. Jone, DyXY – a proximity congestion-aware deadlock-free dynamic routing method for network on chip, in: Proc. of DAC, 2006, pp. 849–852.
[19] G. Ascia et al., Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip, IEEE Trans. Comput. 57 (6) (2008) 809–820.

[20] S. Ma, et al., DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip, in: Proc. of ISCA, 2011, pp. 413–424.
[21] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, H. Tenhunen, Agent-based on-chip network using efficient selection method, in: Proceedings of VLSI-SoC, 2011.
[22] S. Kumar, R. Miikkulainen, Confidence-based Q-routing: an on-line adaptive network routing algorithm, Smart Eng. Syst.: Neural Netw., Fuzzy Logic, Data Min., Evol. Program. 8 (1998) 147–152.
[23] S.P. Choi, D.-Y. Yeung, Predictive Q-routing: a memory-based reinforcement learning approach to adaptive traffic control, in: Advances in Neural Information Processing Systems 8 (NIPS8), 1996, pp. 945–951.
[24] S. Kumar, R. Miikkulainen, Dual reinforcement Q-routing: an on-line adaptive routing algorithm, in: Proc. of the Artificial Neural Networks in Engineering Conference, 1997, pp. 231–238.
[25] M. Majer, et al., Packet routing in dynamically changing networks on chip, in: Proc. of the 19th International Parallel and Distributed Processing Symposium (IPDPS), Denver, CO, USA, 2005.
[26] C. Feng, Z. Lu, A. Jantsch, J. Li, M. Zhang, A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip, in: Proc. of NoCArc, 2010, pp. 11–16.
[27] K.K. Paliwal, J.S. George, N. Rameshan, V. Laxmi, M.S. Gaur, V. Janyani, R. Narasimhan, Implementation of QoS Aware Q-routing Algorithm for Network-on-Chip, 1C3, 2009, pp. 370–380.
[28] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, 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), December 2011, pp. 1–7.
[29] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, J. Plosila, P. Liljeberg, 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), 2012.
[30] M. Ebrahimi, M. Daneshtalab, F. Farahnakian, P. Liljeberg, J. Plosila, M. Palesi, H. Tenhunen, 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), May 2012, pp. 19–26.
[31] M.K. Puthal, V. Singh, M.S. Gaur, V. Laxmi, C-routing: an adaptive hierarchical NoC routing methodology, in: 19th International Conference on VLSI and System-on-Chip, 2011, pp. 392–397.
[32] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, J. Plosila, P. Liljeberg, Optimized Q-learning model for distributing traffic in on-chip networks, in: Proceedings of 3rd IEEE International Conference on Networked Embedded Systems for Every Applications (NESEA), December 2012, pp. 1–8.
[33] A. Varga, et al., The OMNeT++ discrete event simulation system, in: Proc. of the European Simulation Multiconference (ESM'2001), 2001, pp. 319–324.
[34] Y. Ben-Itzhak, E. Zahavi, I. Cidon, A. Kolodny, NoCs simulation framework for OMNeT++, in: Proc. of NOCS, 2011, pp. 265–266.
[35] S.C. Woo, et al., The splash-2 programs: characterization and methodological considerations, in: Proc. of Computer Architecture (ISCA), 1995, pp. 24–36.
[36] M.K. Martin, D.J. Sorin, B.M. Beckmann, et al., Multifacet's general execution driven multiprocessor simulator (GEMS) toolset, SIGARCH Comput. Archit. News 33 (4) (2005) 92–99.

Fahimeh Farahnakian received M.S degree in Computer Engineering from the University of Science and Technology, Tehran, Iran, in 2009. She is currently pursuing her Ph.D. in Embedded Computer and Electronic Systems Laboratory, University of Turku, Finland and from May 2011 she is a doctoral candidate of Graduate School in Electronics, Telecommunications and Automation (GETA). Her research interests include network on- chips, multi-processor system-on-chip, artificial intelligence, machine learning, pattern recognition, fuzzy control, cloud computing and data centers. She is a member of IEEE and is a frequent reviewer for research journals, such as Journal of Circuits, Systems and Computers (JCSC), International Journal of High Performance Systems Architecture (IJHPSA), Journal of Low Power Electronics (JOLPE) and Journal of Supercomputing.

Masoumeh Ebrahimi received her B.S. degree in computer engineering from School of Electrical and Computer Engineering, University of Tehran, Iran in 2005, and M. S. degree in computer architecture from Science and Research University, Iran in 2009. She has received her PhD degree from the University of Turku, Finland in 2013. Currently she is working as a postdoctoral researcher and lecturer at the University of Turku, Finland. Her research interest includes communication protocols, networks-on-chip, 3D integrated systems, systems-on-chip, and neural networks. Masoumeh is a member of IEEE and has published over 60 international peer-refereed journals and conference papers.

Masoud Daneshtalab is an Assistant Professor in Department of Information Technology at University of Turku, Finland. Dr. Daneshtalab is an Associate Editor of World Research Journal of Computer Architecture (JCA) and in the Editorial Board of The Scientific World Journal, International Journal of Distributed Systems and Technologies (IJDST), and International Journal of Embedded and Real-Time Communication Systems (IJERTCS). He has served as a Guest Editor for Springer Computing journal, IET Computers & Digital Techniques, ACM Transactions on Embedded Computing Systems (ACM TECS), and ACM Journal on Emerging Technologies in Computing Systems (JETC), along with Elsevier Journals of Systems Architecture (JSA), Microprocessors and Microsystems (MICPRO), Integration, and Computers & Electrical Engineering. He also co-organizes several special session and workshops including a regular special session on On-Chip Parallel and Network-Based Systems (OCPNBS) in the Euromicro PDP conference, IEEE International workshop on High-Performance Interconnection Networks (HPIN) in conjunction with HPCS, and ACM International workshop on Many-core Embedded Systems (MES) in conjunction with ISCA, and ACM/IEEE International workshop Network-on-Chip Architectures (NoCArc) in conjunction with MICRO. His research interests include on/off-chip interconnection networks, many-core embedded systems, embedded operating systems, FPGA and reconfigurable architectures, 3D stacked architectures, machine learning, and data-centers. He is a member of IEEE and has published 1 book, 3 book chapters, and over 110 refereed international journals and conference papers. He is currently in a Technical Program Committee member of different IEEE and ACM conferences, including NOCS, DATE, ASPDAC, ESTIMedia, VLSI Design, SOCC, DSD, PDP, EmbeddedCom, ICESS, EUC, NESEA, CASEMANS, NoCArc, MES, HPIN, and JEC-ECC.

Pasi Liljeberg received his M.Sc. and Ph.D. degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively. He is an Associate Professor in Embedded Electronics laboratory and an Adjunct Professor in embedded computing architectures at the University of Turku, Embedded Computer Systems laboratory. During the period 2007–2009 he held an Academy of Finland researcher position. Adj.Prof. Liljeberg is the author of over 150 peer- reviewed publications, has supervised 9 PhD theses. His current research interests include parallel and distributed systems, Internet-of-Things, embedded computing architecture, fault tolerant and energy aware system design, 3D multiprocessor system architectures, dynamic power management, cyber physical systems, intelligent network-on-chip communication architectures and reconfigurable system design. He has established a research group focusing on reliable and fault tolerant self-timed communication platforms for multiprocessor systems, FastCop project, 2008–2011, Academy of Finland.

Juha Plosila is an Associate Professor in Embedded Computing and an Adjunct Professor in Digital Systems Design at the University of Turku (UTU), Department of Information Technology, Finland. He received a PhD degree in Electronics and Communication Technology from UTU in 1999. Dr. Plosila is the leader of the Embedded Computer and Electronic Systems (ECES) research unit and a co-leader of the Resilient IT Infrastructures (RITES) research programmer at Turku Centre for Computer Science (TUCS). He leads the Embedded Systems master's program at the EIT ICT Labs Master School and is a management committee member of the EU COST Actions IC1103 (MEDIAN: Manufacturable and Dependable Multicore Architectures at Nanoscale) and IC1202 (TACLe: Timing Analysis on Code Level). Dr. Plosila is an Associate Editor of International Journal of Embedded and Real-Time Communication Systems (IGI Global). His current research deals with adaptive multiprocessor systems at different abstraction levels. This includes e.g. specification, development, and verification of self-aware, multi-agent monitoring and control architectures for massively parallel systems, as well as applications of autonomous energy-efficient architectures to new computational challenges in the cyber-physical systems domain.