J Supercomput (2012) 59:882–900
DOI 10.1007/s11227-010-0477-0

Tree-turn routing: an efficient deadlock-free routing algorithm for irregular networks

Jiazheng Zhou · Yeh-Ching Chung

Published online: 9 September 2010

Abstract In this paper, we propose a general turn model, called a *Tree-turn* model, for tree-based routing algorithms on irregular topologies. In the *Tree-turn* model, links are classified as either a *tree link* or a *cross link* and six directions are associated with the channels of links. Then we can prohibit some of the turns formed by these six directions such that an efficient deadlock-free routing algorithm, *Tree-turn* routing, can be derived. There are three phases to develop the *Tree-turn* routing. First, a coordinated tree for a given topology is created. Second, a communication graph is constructed based on the topology and the corresponding coordinated tree. Third, the forwarding table is set up by using all-pairs shortest path algorithm according to the prohibited turns in the *Tree-turn* model and the directions of the channels in the communication graph. To evaluate the performance of the proposed *Tree-turn* routing, we develop a simulator and implement *Tree-turn* routing along with *up*/down** routing, *L-turn* routing, and *up*/down** routing with DFS methodology. The simulation results show that *Tree-turn* routing outperforms other routing algorithms for all the test cases.

Keywords Tree-turn model · Tree-turn routing · Deadlock-free · Irregular networks

1 Introduction

Supercomputers can be made of mainframes or network-based distributed systems like clusters. Nowadays, the trend is towards cost-effective network-based distributed

J. Zhou · Y.-C. Chung (✉)
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, ROC
e-mail: ychung@cs.nthu.edu.tw

J. Zhou
e-mail: jzzhou@cs.nthu.edu.tw

computing systems since they consist of commodity components, such as personal computers and high speed networks [1, 14, 24].

Routing on regular topology can be elaborately designed and achieve good performance such as XY routing for mesh and MLID routing [20] for a fat-tree topology. However, a regular topology may become an irregular topology due to the failure of components. To deliver routing algorithms on an irregular topology, connectivity and deadlock-free properties must be guaranteed. Connectivity can be solved by using tree-based routing algorithms, to build a spanning tree of the topology to connect all the nodes. Deadlock occurs when nodes are involved in cyclic waiting of messages, for example, in a wormhole switching network [23]. To avoid deadlock, one method is to get rid of the possibility to form a cycle. Many methods to prevent deadlocks have been proposed [7, 8, 11, 13, 19, 29]. Among them, the turn model proposed in [11, 13] is a tool to deliver partially adaptive routing algorithms. It analyzes the directions of messages and prohibits enough turns to break the turn cycles to avoid deadlocks.

Several tree-based routing algorithms based on the turn model have been proposed for irregular topologies. In a tree-based routing algorithm, a spanning tree is first formed from a given irregular topology. Based on the spanning tree, each node can be assigned a coordinate. From the coordinates of the nodes, each directed link (channel) can be assigned a direction. Two directions can be used to form a turn. By carefully determining which turns to prohibit in the turn space, one is then able to break the turn cycle and therefore avoid deadlocks. *Up*/down** routing [28] is the first tree-based routing algorithm developed for an irregular topology. Based on 1D turn model, *up*/down** routing provides only two directions, up and down, associated with its channels. A legal route of *up*/down** routing follows the rule: a packet must traverse zero or more links in the up direction followed by zero or more links in the down direction. From the turn model point of view, turn $T_{down,up}$ is prohibited in *up*/down** routing. Although the *up*/down** routing is simple, its performance is not good due to the unbalanced traffic. Since there are only two directions in *up*/down** routing, there is not much flexibility when selecting prohibited turns.

To overcome the drawbacks of *up*/down* routing*, *L-turn routing* is proposed [17] based on the 2D turn model [15–17]. In *L-turn* routing, there are four directions, left-up, left-down, right-up, and right-down, associated with each channel. By carefully setting up the prohibited turns for each node, one can obtain a more even distribution of traffic load and shorter routing paths compared to the *up*/down** routing. Sancho et al. [27] tried to improve the *up*/down** routing by using a DFS methodology. They built a DFS spanning tree instead of a BFS spanning tree used originally in *up*/down** routing. *Up*/down** routing with DFS methodology has less restrictions (prohibited turns) compared to the original *up*/down** routing. There are also some other routings for irregular networks such as smart routing [2], flexible (FX) routing [26], segment-based routing (SR) [22], descending layers (DL) routing [18], layered (LASH) routing [21], and LASH-TOR routing [32]. Among these routings mentioned above, only *up*/down** routing, *L-turn* routing, smart routing, flexible routing, and segment-based routing can work without virtual channels. Virtual channels [4, 6] can be used to solve the cycle dependency issue to design a deadlock-free routing or to be used to provide quality of service. However, not all of the interconnection networks provide virtual channels; for example, Myrinet does not implement virtual channels. Therefore, we need to provide an efficient routing algorithm that does not rely on virtual channels and can be applied to all kinds of interconnection networks.

In this paper, we first propose a turn model, *Tree-turn* model, for tree-based routing algorithms on irregular topologies. We intend to explore the turn space and find a better routing algorithm based on the turn model. In *Tree-turn* model, the directions of channels can be classified into left-up, left, left-down, right-up, right, and right-down directions. The *Tree-turn* model has two more directions, left and right, than the 2D turn model. In addition, tree links and cross links are associated with different channels of links (directions). Tree links can only have left-up and right-down directions and cross links have left, left-down, right-up, and right directions. By carefully selecting prohibited turns, we can push the traffic downward in a spanning tree get more balanced traffic.

To evaluate the performance of the four *Tree-turn* routing algorithms, we compare them with *up*/down** routing, *L-turn* routing, and *up*/down** routing with DFS methodology. In experimental test, we implement a simulator for these routing algorithms. The simulation results show that *Tree-turn* routing outperforms other routing algorithms for irregular networks.

The rest of the paper is organized as follows. We introduce some routing algorithms for switching networks in Sect. 2. The definitions and terms used in this paper are given in Sect. 3. In Sect. 4, we describe the *Tree-turn* model in detail. The *Tree-turn* routings derived from *Tree-turn* model are given in Sect. 5. In Sect. 6, we show the experimental test of *Tree-turn* routings along with *up*/down** routing, *L-turn* routing, and *up*/down** routing with DFS methodology. The conclusions and future work are given in Sect. 7.

2 Related work

The turn model proposed by Glass and Ni [11, 12] is a model for designing deadlock-free [5] algorithms. Based on the turn model, Glass and Ni propose three partially adaptive routing algorithms, *west-first* routing, *north-last* routing, and *negative-first* routing, for 2D meshes. They also extend the algorithms to n-dimensional networks such as n-dimensional meshes and k-ary n-cubes.

Up/down** routing was first introduced in Autonet [28]. It is the most popular tree-based routing algorithm. The routing algorithm is simple and there are only two directions to assign to the links. However, there are more restrictions in the network and therefore the performance is not good. *L-turn* routing [15–17] introduces 2D turn model to improve the performance. It explores the turn space and obtains an average distance that is shorter than that of *up*/down** routing. However, in *L-turn* routing, the tree links (edges in a spanning tree) and the cross links (edges not in the spanning tree) are considered to be the same type of link. When we explore the turn space, we can consider these links as two different types. Since tree links are links in the spanning tree, it is possible for one node in the network to communicate with any other node by just using the tree links. The cross links, in contrast, are able to provide more than one path between two nodes, which can be used as a way to relieve the congested traffic in the spanning tree.

Up/down** routing with DFS methodology [27] uses a DFS spanning tree on the network graph instead of BFS spanning tree. By using this method, the number of routing restrictions can be decreased to improve the performance. The authors also propose several heuristic rules for computing the spanning trees. Segment-based routing [22] is a fault-tolerant routing for meshes and tori. It partitions a topology into subnets, and subnets into segments. They have three types of segments, with each segment having a specified routing restriction. The purpose of the routing restrictions is to ensure that there is deadlock-freedom and to preserve connectivity. The segments must be made as short as possible and the routing restrictions must be carefully placed. Otherwise, the performance of segment-based routing may be worse than that of *up*/down** routing.

In the paper by Koibuchi et al., descending layers routing is proposed [18]. In this deadlock-free deterministic routing algorithm, the network is divided into several layers of sub-networks with the same topology using virtual channels. Each sub-network is able to use one of various deadlock-free routings. Moreover, no cyclic dependency is formed across the sub-networks since between the packets the sub-networks are passed in the descending order. Lysne et al. [21, 32] also take advantage of virtual channels and propose layered routing. They group virtual channels into network layers and have each layer assigned a limited number of source/destination pairs. By separating the traffic, they are able to improve the performance.

The odd–even turn model [3] is proposed for adaptive routing in meshes. It is a novel model since it restricts the locations where some types of turns can be taken instead of just prohibiting turns to break the cycles. A fault-tolerant protocol based on odd-even turn model is proposed in [33].

There are some other routing algorithms that improve the performance of *up*/down** routing. In [25, 31], the proposed adaptive routing algorithms are better than *up*/down** routing. In [9, 10, 30], the authors use the virtual channel concept to reduce the latency of the *up*/down** routing.

3 Preliminaries

In this section, we will give definitions and terms used in this paper.

Definition 1 (Graph) A switch-based network can be represented as a graph $G = (V, E)$, where V is the set of the switches, E is the set of the bidirectional links between switches, and G is the network topology. For link $e = (v_i, v_j)$ in E, there are two communication channels $\langle v_i, v_j\rangle$ and $\langle v_j, v_i\rangle$ such that node v_i can send a message to node v_j through $\langle v_i, v_j\rangle$ and node v_j can send a message to node v_i through $\langle v_j, v_i\rangle$. For a channel $\langle v_i, v_j >$, v_i and v_j are called *start* and *sink* nodes of the channel, respectively. $\langle v_i, v_j\rangle$ is called the *output* channel of v_i and *input* channel of v_j.

Definition 2 (Coordinated tree) Given $G = (V, E)$, a *coordinated tree* (CT) is a breadth-first search (BFS) or depth-first search (DFS) spanning tree of G, where $CT = (V, E')$ and $E' \subseteq E$. For each node v in a coordinated tree, node v is associated with a two-dimensional coordinate $v(x, y)$. We use $X(v)$ and $Y(v)$ to denote the x and y coordinates of node v, respectively, that is, $X(v) = x$ and $Y(v) = y$. $Y(v)$ is defined as the level of node v in the coordinated tree, and $X(v)$ is defined as the order of preorder traversal of the coordinated tree starting from the root to node v.

Since two or more children nodes can be selected as the next preorder traversal node, it is possible to build several different coordinated trees using the same network topology. To obtain a unique coordinated tree for a given network topology, the node with smaller network ID is selected before the node with a larger network ID when performing the preorder traversal.

Definition 3 (Tree link and cross link) Given $G = (V, E)$ and a coordinated tree $CT = (V, E')$ of G, E' and $E - E'$ are the sets of *tree links* and *cross links* of G with respect to CT, respectively.

Definition 4 (Communication graph (CG)) Given $G = (V, E)$ and a coordinated tree $CT = (V, E')$ of G, the *communication graph* $CG = (V, \vec{E})$ is a directed graph with respect to G and CT, where $\vec{E}$ is the set of all communication channels of E.

Definition 5 (Direction) Given a communication graph $CG = (V, \vec{E})$, for each channel $\vec{e} = \langle v_i, v_j \rangle \in \vec{E}$, we define

(1) v_j is the *left-up* node of v_i if $X(v_j) < X(v_i)$ and $Y(v_j) < Y(v_i)$.
(2) v_j is the *left* node of v_i if $X(v_j) < X(v_i)$ and $Y(v_j) = Y(v_i)$.
(3) v_j is the *left-down* node of v_i if $X(v_j) < X(v_i)$ and $Y(v_j) > Y(v_i)$.
(4) v_j is the *right-up* node of v_i if $X(v_j) > X(v_i)$ and $Y(v_j) < Y(v_i)$.
(5) v_j is the *right* node of v_i if $X(v_j) > X(v_i)$ and $Y(v_j) = Y(v_i)$.
(6) v_j is the *right-down* node of v_i if $X(v_j) > X(v_i)$ and $Y(v_j) > Y(v_i)$.

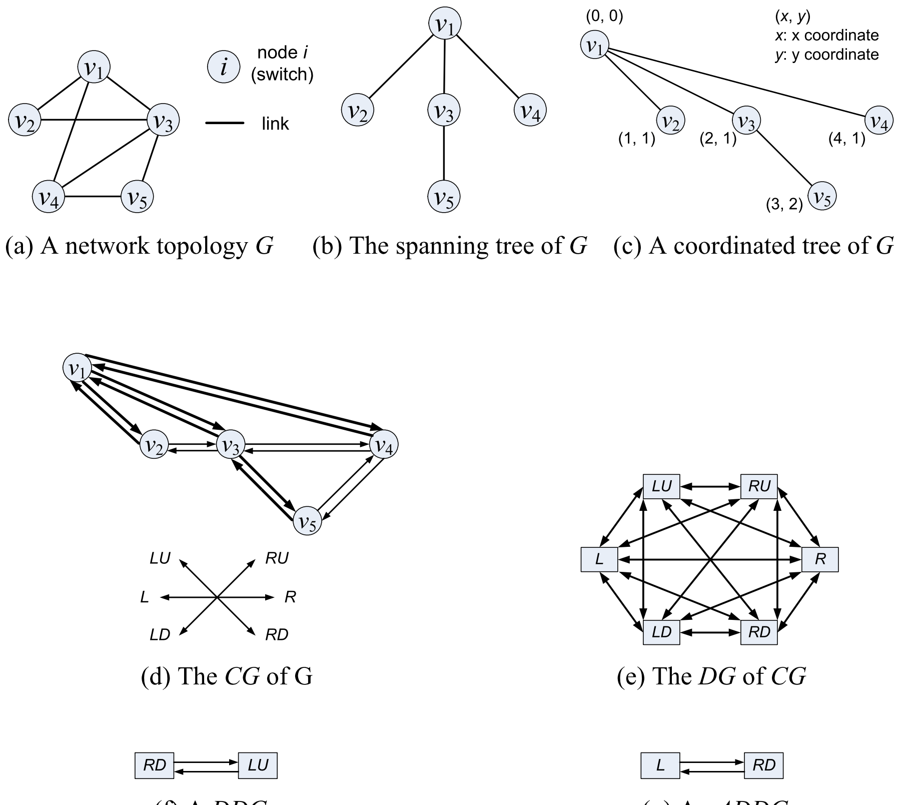
For each channel $\vec{e} = \langle v_i, v_j \rangle$, the direction of $\vec{e}$, denoted as $d(\vec{e})$, is defined as *LU*, *L*, *LD*, *RU*, *R*, and *RD* if v_j is the *left-up* node, the *left* node, the *left-down* node, the *right-up* node, the *right* node, and the *right-down* node of v_i, respectively.

Definition 6 (Turn) Given a communication graph $CG = (V, \vec{E})$, the directions of $\vec{e}_\alpha$ and $\vec{e}_\beta$ form a *turn* for v_j if $\vec{e}_\alpha = \langle v_i, v_j \rangle$ and $\vec{e}_\beta = \langle v_j, v_k \rangle$. We use $T_{d(\vec{e}_\alpha), d(\vec{e}_\beta)}$ to denote the turn formed by the directions of $\vec{e}_\alpha$ and $\vec{e}_\beta$.

Definition 7 (Turn cycle) Given a communication graph $CG = (V, \vec{E})$, a *turn cycle* $TC = (T_{d(\vec{e}_1), d(\vec{e}_2)}, T_{d(\vec{e}_2), d(\vec{e}_3)}, \ldots, T_{d(\vec{e}_k), d(\vec{e}_{k+1})})$ is a sequence of turns in which the sink node of the first channel is also the sink node of the last channel in the turn sequence, that is, the start node of $\vec{e}_2$ is the sink node of $\vec{e}_{k+1}$.

Definition 8 (Direction graph (DG)) The *direction graph* $DG = (D, \vec{T})$ with respect to a communication graph $CG = (V, \vec{E})$ is a complete directed graph, where D is the set of directions defined in CG and $\vec{T} = \{T_{d_i, d_j} \mid \text{for all } d_i, d_j \in D \text{ and } d_i \neq d_j\}$ is the set of all possible turns that can be defined in CG. A DG is called the *complete direction graph* (*CDG*) if $D = \{LU, L, LD, RU, R, RD\}$.

Definition 9 (Direction dependency graph (DDG)) Given a *DG*, any subset of *DG* is defined as the *direction dependency graph* (*DDG*) of *DG*.



(a) A network topology *G* (b) The spanning tree of *G* (c) A coordinated tree of *G*

(d) The *CG* of G (e) The *DG* of *CG*

(f) A *DDG* (g) An *ADDG*

Fig. 1 Examples of the various definitions

Definition 10 (Acyclic direction dependency graph (ADDG)) Given a *CG*, the *DG* of *CG*, and a *DDG* of *DG*, for each node v in *CG*, if the edges of *DDG* are the only available turns allowed at v and no turn cycle can be formed in *CG*, then the *DDG* is called *acyclic DDG*.

Definition 11 (Maximal acyclic direction dependency graph (Maximal ADDG)) Given a *CG*, the *DG* of *CG*, an *ADDG* of *DG* is called the *maximal ADDG* if adding any edge that in *DG* but not in *ADDG* to the *ADDG* will result in turn cycles in *CG*.

We now give an example to explain the above definitions. In Fig. 1(a), we use a graph $G = (V, E)$ to represent a switched-based network, where $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_3, v_4), (v_3, v_5), (v_4, v_5)\}$. In Fig. 1(b), a BFS spanning tree of the network in Fig. 1(a) is shown. The root in the BFS spanning tree is node v_1. The *coordinated tree* of G is shown in Fig. 1(c). In Fig. 1(c), according to Definition 2, we have $Y(v_1) = 0$, $Y(v_2) = 1$, $Y(v_3) = 1$, $Y(v_4) = 1$, and $Y(v_5) = 2$. When performing preorder traversal, we have $X(v_1) = 0$. Nodes v_2, v_3, v_5, and v_4 are traversed in order if we choose the node with smaller ID as the next node. We have $X(v_2) = 1$, $X(v_3) = 2$, $X(v_5) = 3$, and $X(v_4) = 4$. Node v_3 is the *right-down*, *right*, *left*, and *left-up* node of nodes v_1, v_2, v_4, and v_5, respectively.

Figure 1(d) shows the *communication graph* of Fig. 1(a) and Fig. 1(c). We use thick links and thin links to represent tree links and cross links in Fig. 1(d), respectively. In Fig. 1(d), the directions $d(\langle v_1, v_2\rangle) = RD$, $d(\langle v_2, v_1\rangle) = LU$, $d(\langle v_2, v_3\rangle) = R$, $d(\langle v_3, v_2\rangle) = L$, $d(\langle v_4, v_5\rangle) = LD$, and $d(\langle v_5, v_4\rangle) = RU$. We can see that the directions of tree links are either *LU* or *RD*, and the directions of cross links are *L*, *LD*, *RU*, or *R*. $T_{d(\langle v_2,v_1\rangle),d(\langle v_1,v_3\rangle)} = T_{LU,RD}$ is a *turn* and $TC = \{T_{d(\langle v_2,v_1\rangle),d(\langle v_1,v_3\rangle)}, T_{d(\langle v_1,v_3\rangle),d(\langle v_3,v_2\rangle)}, T_{d(\langle v_3,v_2\rangle),d(\langle v_2,v_1\rangle)}\} = \{T_{LU,RD}, T_{RD,L}, T_{L,LU}\}$ is a *turn cycle*.

In Fig. 1(e), the *direction graph DG* of Fig. 1(d) is shown. It is a complete direction graph since it consists of six directions. Figure 1(f) shows a *direction dependency graph DDG* of Fig. 1(e). There are two turns $T_{RD,LU}$ and $T_{LU,RD}$ in the *DDG*. Turn cycles $\{T_{RD,LU}, T_{LU,RD}\}$ and $\{T_{LU,RD}, T_{RD,LU}\}$ are formed in the *DDG*. Figure 1(g) shows an *acyclic direction dependency graph ADDG* of Fig. 1(e). It has two turns $T_{L,RD}$ and $T_{RD,L}$. If we only allow these two turns in Fig. 1(d), the two turns form a cycle but not a turn cycle. We can see that a cycle in an *ADDG* will not result in a turn cycle in *CG*.

4 The *Tree-turn* model

Given an irregular topology G, based on Definitions 2, 3, 4, and 5, the directions of channels can be classified into six directions, *LU*, *L*, *LD*, *RU*, *R*, and *RD*, in *Tree-turn* model. The *Tree-turn* model has two more directions, *L* and *R*, than the 2D turn model. In addition, since the coordinated tree of G is skewed and we define tree links as the links of the coordinated tree, for each channel $\vec{e}$ in tree links, the direction of $\vec{e}$ is either *LU* or *RD*, that is, $d(\vec{e}) \in \{LU, RD\}$. For each channel $\vec{e}$ in cross links, the direction of $\vec{e}$ is *L*, *LD*, *RU*, or *R*, that is, $d(\vec{e}) \in \{L, LD, RU, R\}$. Tree links and cross links are associated with different directions in the *Tree-turn* model. By association with directions to tree links and cross links, we can use cross links to push the traffic downward in a spanning tree and reduce the congested traffic of the spanning tree.

In order to avoid deadlocks, in the *Tree-turn* model, a maximal *ADDG* is derived from the *CDG* that contains six directions. Since no turn cycle can be formed in a maximal *ADDG* and the *DG* of a topology G contains at most six directions, when applying the prohibited turns derived from the maximal *ADDG* of the *CDG* to nodes of G, a deadlock-free routing can be preserved. There are two issues to find the maximal *ADDG* from the *CDG*. The first issue is to decide what edges should be removed (prohibited) from the *CDG*. The second issue is the routing algorithm derived from the found maximal *ADDG* should it perform efficiently. For the first issue, we use an incremental method to remove edges step by step from the *CDG* to obtain a maximal *ADDG*. For the second issue, to get more balanced traffic, we will try to prevent the traffic from flowing to the root of a *CG* and push the traffic flow downward to the leaves of a *CG*; that is, we will remove edges that will make traffic flow upward and

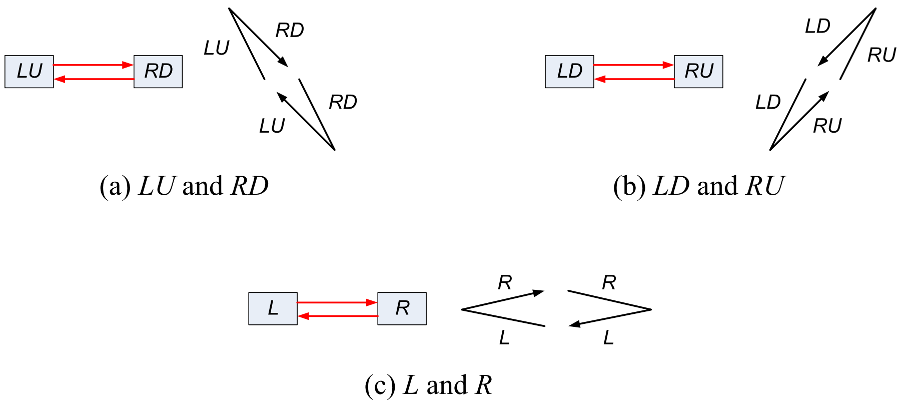


Fig. 2 The *DG*s of node pairs and their possible turn cycles

keep edges that will push the traffic flow downward as much as possible. The process of finding the maximal *ADDG* from the *CDG* consists of the following three steps:

Step 1 Find the maximal *ADDG*s: $ADDG_1$, $ADDG_2$, and $ADDG_3$ from *DG*s of nodes *LU* and *RD*, nodes *LD* and *RU*, and nodes *L* and *R* from the *CDG*, respectively.

Step 2 Combine $ADDG_1$ with $ADDG_2$ by adding edges between nodes in $ADDG_1$ and $ADDG_2$ to form a new *DDG* and find a maximal *ADDG*, $ADDG_4$, from the new formed *DDG*.

Step 3 Combine $ADDG_3$ with $ADDG_4$ by adding edges between nodes in $ADDG_3$ and $ADDG_4$ to form a new *DDG* and find a maximal *ADDG*, $ADDG_5$, from the new formed *DDG*. The found $ADDG_5$ is a maximal *ADDG* of the *CDG*.

In the following, we will describe these three steps in detail.

4.1 Step 1

In this step, we will find the maximal *ADDG*s: $ADDG_1$, $ADDG_2$, and $ADDG_3$ from *DG*s of nodes *LU* and *RD*, nodes *LD* and *RU*, and nodes *L* and *R* from the *CDG*, respectively. The reason we choose these node pairs is that the *DG* of each node pair contains edges with opposite directions. These edges form a cycle that may lead to a turn cycle. Figure 2 shows the *DG*s of these node pairs and their corresponding possible turn cycles.

To prevent the cycles of *DG*s shown in Fig. 2, we must remove one edge from each *DG*. In Fig. 2(a), we remove the edge $T_{RD,LU}$ since this is the only choice. The reason this is done is to maintain the connectivity of the topology. Since the *LU* and *RD* directions are defined for tree links, should the topology be a tree and an edge $T_{LU,RD}$ get removed, there is no way for all nodes to communicate with each other. This is because the tree is now broken, and there will now be at least one leaf node that is unable to communicate with the root for example. By removing edge $T_{RD,LU}$ from Fig. 2(a), we can get $ADDG_1$ shown in Fig. 3(a). In Fig. 2(b), we can break the cycle by removing either edge of the *DG*. For each node v in the *CG*, the direction

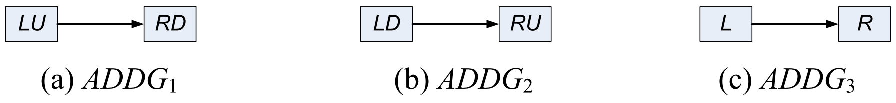


Fig. 3 The maximal *ADDG*s of *DG*s shown in Fig. 2

LD means that the traffic flow is going downward from node v to other nodes whose y coordinate is less than that of node v. Edge $T_{LD,RU}$ means that the traffic flow is going downward before going upward. In order to push traffic downward, we keep edge $T_{LD,RU}$. By removing edge $T_{RU,LD}$ from Fig. 2(b), we can get $ADDG_2$ shown in Fig. 3(b). In Fig. 2(c), the cycle is formed by directions L and R. Since it does not affect the traffic flow going downward or upward by removing either edge, we remove edge $T_{R,L}$ in this case. By removing edge $T_{R,L}$ from Fig. 2(c), we can get $ADDG_3$ shown in Fig. 3(c).

4.2 Step 2

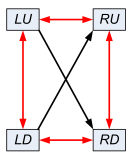
In this step, we want to combine $ADDG_1$ with $ADDG_2$ by adding edges between nodes in $ADDG_1$ with $ADDG_2$ to form a new *DDG* and find $ADDG_4$ from the new formed *DDG*. The *DDG* by combining $ADDG_1$ with $ADDG_2$ is shown in Fig. 4(a). In Fig. 4(a), there are four cycles C_1, C_2, C_3, and C_4 that will result in turn cycles $TC_1 = \{T_{RD,RU}, T_{RU,LU}, T_{LU,RD}\}$, $TC_2 = \{T_{LD,RU}, T_{RU,LU}, T_{LU,LD}\}$, $TC_3 = \{T_{RU,RD}, T_{RD,LD}, T_{LD,RU}\}$, and $TC_4 = \{T_{RD,LD}, T_{LD,LU}, T_{LU,RD}\}$ in a *CG* as shown in (b), (c), (d), and (e) of Fig. 4, respectively. To break these four turn cycles, we need to remove some edges from the *DDG* shown in Fig. 4(a).

Both cycles C_1 and C_2, have a common edge $T_{RU,LU}$. This particular edge makes the traffic flow upward. In order to push the traffic flow downward to the leaves of a corresponding *CT*, we remove this common edge and break cycles C_1 and C_2. For cycles C_3 and C_4, they have a common edge $T_{RD,LD}$. Since edge $T_{RD,LD}$ makes the traffic flow downward, we keep the edge. For other edges $T_{RU,RD}$ and $T_{LD,RU}$ in cycle C_3, $T_{RU,RD}$ makes the traffic flow upward then downward and $T_{LD,RU}$ makes the traffic flow downward then upward. In order to push the traffic flow downward to leaves of a corresponding *CT*, we remove edge $T_{RU,RD}$ to break cycle C_3. For other edges $T_{LU,RD}$ and $T_{LD,LU}$ in cycle C_4, since *LU* and *RD* are directions of tree links, we cannot remove $T_{LU,RD}$ for the reason of connectivity as stated in Step 1. Therefore, we remove the edge $T_{LD,LU}$ to break cycle C_4. We then obtain the $ADDG_4$ as shown in Fig. 4(f).

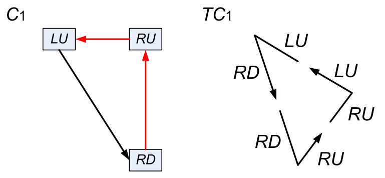
4.3 Step 3

In this step, we want to combine $ADDG_3$ with $ADDG_4$ by adding edges between nodes in $ADDG_3$ and $ADDG_4$ to form a new *DDG* and find $ADDG_5$ from the newly formed *DDG*. For nodes in Fig. 4(f), we have the following observations:

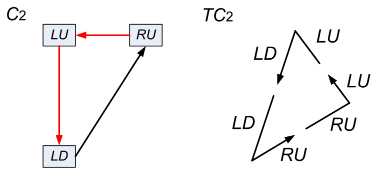
Observation 1 No combination of edges from nodes *LD* and *RD* will have an upward direction in a *CG*.



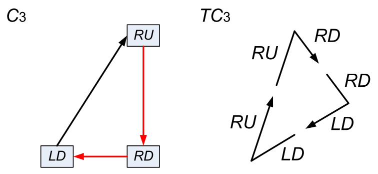
(a) The *DDG* by combining $ADDG_1$ with $ADDG_2$



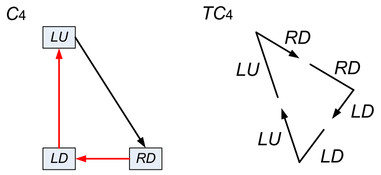
(b) Cycle C_1 and turn cycle TC_1



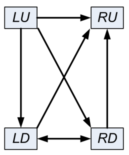
(c) Cycle C_2 and turn cycle TC_2



(d) Cycle C_3 and turn cycle TC_3



(e) Cycle C_4 and turn cycle TC_4



(f) $ADDG_4$

Fig. 4 Combine $ADDG_1$ with $ADDG_2$ to form $ADDG_4$

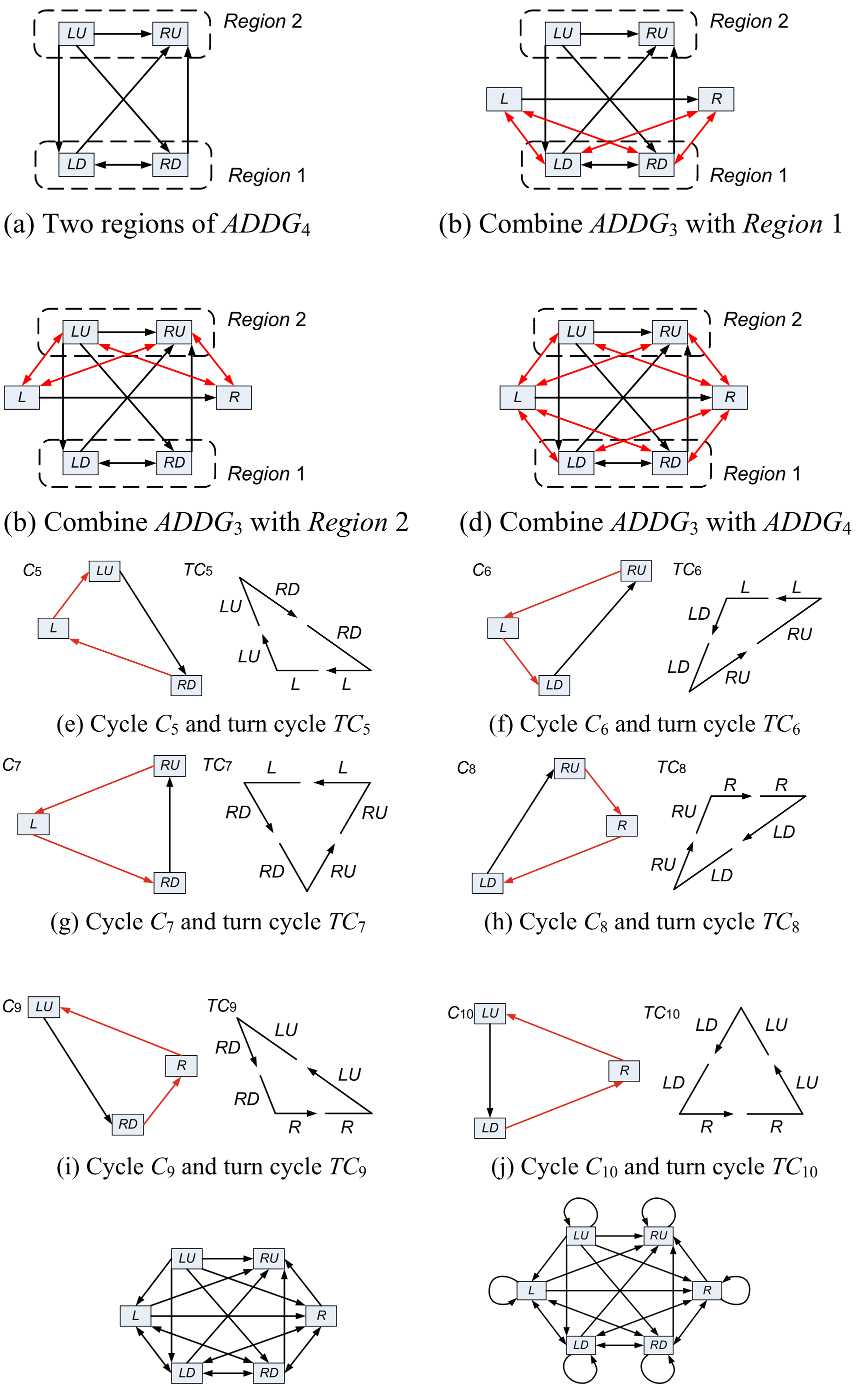
Observation 2 No combination of edges from nodes *LU* and *RU* will have a downward direction in a *CG*.

Therefore, we divide $ADDG_4$ into *Region* 1 and *Region* 2 as shown in Fig. 5(a). For the $ADDG_3$ shown in Fig. 3(c), edge $T_{L,R}$ indicates that the traffic is flowing between nodes in the same level of a corresponding *CT*. When we combine $ADDG_3$ with *Region* 1 or *Region* 2 shown in Fig. 5(a), we have the following observations:

Observation 3 If we combine $ADDG_3$ with *Region* 1 to form a *DDG* shown in Fig. 5(b), no turn cycles can be formed by applying edges of the *DDG* to nodes of a given *CG*.

Observation 4 If we combine $ADDG_3$ with *Region* 2 to form a *DDG* shown in Fig. 5(c), no turn cycles can be formed by applying edges of the *DDG* to nodes of a given *CG*.

Observation 5 If we combine $ADDG_3$ with $ADDG_4$, there are two possible ways to form turn cycles. One way is to traverse from node v in $ADDG_3$ to nodes in *Region* 1, then to nodes in *Region* 2, and then to return to node v. The other way is to traverse



(a) Two regions of *ADDG*$_4$ (b) Combine *ADDG*$_3$ with *Region* 1

(b) Combine *ADDG*$_3$ with *Region* 2 (d) Combine *ADDG*$_3$ with *ADDG*$_4$

(e) Cycle C_5 and turn cycle TC_5 (f) Cycle C_6 and turn cycle TC_6

(g) Cycle C_7 and turn cycle TC_7 (h) Cycle C_8 and turn cycle TC_8

(i) Cycle C_9 and turn cycle TC_9 (j) Cycle C_{10} and turn cycle TC_{10}

(k) *ADDG*$_5$ (l) The maximal *ADDG* of *Tree-turn* model

Fig. 5 Combine *ADDG*$_3$ with *ADDG*$_4$ to form *ADDG*$_5$

from node v in $ADDG_3$ to nodes in *Region* 2, then to nodes in *Region* 1, and then to return to node v.

Based on Observations 3, 4, and 5, in Fig. 5(d), there are six cycles C_5, C_6, C_7, C_8, C_9, and C_{10} that will result in turn cycles $TC_5 = \{T_{L,LU}, T_{LU,RD}, T_{RD,L}\}$, $TC_6 = \{T_{L,LD}, T_{LD,RU}, T_{RU,L}\}$, $TC_7 = \{T_{L,RD}, T_{RD,RU}, T_{RU,L}\}$, $TC_8 = \{T_{R,LD}, T_{LD,RU}, T_{RU,R}\}$, $TC_9 = \{T_{R,LU}, T_{LU,RD}, T_{RD,R}\}$, and $TC_{10} = \{T_{R,LU}, T_{LU,LD}, T_{LD,R}\}$ as shown in (e), (f), (g), (h), (i), and (j) of Fig. 5, respectively.

For cycle C_5, edges $T_{L,LU}$ and $T_{LU,RD}$ make the traffic flow upward. Since *LU* and *RD* are directions of tree links, we cannot remove $T_{LU,RD}$ for the reason of connectivity. In order to push the traffic flow downward to the leaves of a corresponding *CT*, we remove the edge $T_{L,LU}$ to break the cycle C_5. For cycles C_6 and C_7, there is a common edge $T_{RU,L}$ that makes the traffic flow upward. In order to push the traffic flow downward to the leaves of a corresponding *CT*, we remove the edge $T_{RU,L}$ in order to break cycles C_6, and C_7. For cycle C_8, since the only edge $T_{RU,R}$ makes the traffic flow upward ($T_{LD,RU}$ makes the traffic flow downward instead of upward), in order to push the traffic flow downward to leaves of a corresponding *CT*, we remove the edge $T_{RU,R}$ to break cycle C_8. For cycles C_9 and C_{10}, there is a common edge $T_{R,LU}$ that makes the traffic flow upward. In order to push the traffic flow downward to the leaves of a corresponding *CT*, we remove the edge $T_{R,LU}$ to break cycles C_9, and C_{10}. We now obtain $ADDG_5$ as shown in Fig. 5(k). Since turns that travel in the same direction are permitted, the maximal *ADDG* of *Tree-turn* model is shown in Fig. 5(l).

From Step 1 to Step 3, we have removed 10 edges from *CDG*. These removed edges are prohibited turns, denoted as $PT = \{T_{L,LU}, T_{LD,LU}, T_{RU,LU}, T_{R,LU}, T_{RD,LU}, T_{RU,L}, T_{R,L}, T_{RU,LD}, T_{RU,R}, T_{RU,RD}\}$, in *Tree-turn* model.

5 The *Tree-turn* routing

Based on the *Tree-turn* model, given an irregular topology $G = (V, E)$, we are able to derive the corresponding *Tree-turn* routing by the following three phases:

Phase 1 Construct the corresponding coordinated tree $CT = (V, E')$ of G.
Phase 2 Construct the communication graph $CG = (V, \vec{E})$ from G and CT.
Phase 3 Set up the forwarding tables of nodes in CG by using the all-pairs shortest path algorithm according to the 10 prohibited turns derived from *Tree-turn* model and the directions of channels in CG.

In phase 1, we use *upper-channel first* [15] to choose the next-visit node when we construct the coordinated tree. In phase 2, we use G and CT from the previous phase to construct the communication graph $CG = (V, \vec{E})$. In phase 3, for the all-pairs shortest path algorithm, whenever we find a shorter routing path through node k and the turn formed at node k is not a prohibited turn in the *Tree-turn* model, we adjust the routing path and set up the forwarding tables of the nodes on the routing path. Otherwise, we will keep the original routing path. The algorithm of setting up forwarding tables is shown in Fig. 6.

```
Algorithm set_up_forwarding_tables()
1. Let $n$ be the number of nodes in the network.
2. Let $routing\_path[i][j]$ be the routing path from node $i$ to node $j$.
3. Let $length[i][j]$ be the length from node $i$ to node $j$.
4. Let $direction(i, j)$ be the direction of channel $\langle i, j\rangle$ formed by node $i$ and node $j$.
5. Let $turn(d_i, d_j)$ be the turn form direction $d_i$ and direction $d_j$.
6. /* Initialize the $length[i][j]$ according to the adjacency matrix. */
     for $i = 1$ to $n$ do
       for $j = 1$ to $n$ do
         /* Initialization */
         $length[i][j] = \infty$; $routing\_path[i][j] = $ NULL;
         if (there exists one link between node $i$ and node $j$) then
           $length[i][j] = 1$;
           Append node $j$ to $routing\_path[i][j]$.
         end_if
       end_for
     end_for
7. /* Compute the $length[i][j]$ and adjust the routing paths. */
     for $k = 1$ to $n$ do
       for $i = 1$ to $n$ do
         for $j = 1$ to $n$ do
           /* If we can find a shorter path through node $k$. $*$ /
           /* We also consider the paths with the same length. */
           if ($length[i][k] + length[k][j] < length[i][j]$) then
             Let node $x$ be the ($length[i][k] - 1$)-th node of $routing\_path[i][k]$.
             Let node $y$ be the first node of $routing\_path[k][j]$.
             $inDirection = direction(x, k)$;
             $outDirection = direction(k, y)$;
             if ($turn(inDirection, outDirection)$ is not prohibited) then
               $length[i][j] = length[i][k] + length[k][j]$;
               $routing\_path[i][j] = routing\_path[i][k] + routing\_path[k][j]$;
               Set up the forwarding tables for the nodes on the routing paths.
             end_if
           end_if
         end_for
       end_for
     end_for
End_Algorithm
```

Fig. 6 The algorithm of setting up forwarding tables

Theorem 1 *The Tree-turn routing is deadlock-free and there exists at least one path from one node to another in a CG.*

Proof Based on the *Tree-turn* model, there is at least one prohibited turn to break each turn cycle in the *CDG*. Therefore, this routing algorithm is deadlock-free. Since the *turn* $T_{LU,RD}$ is not prohibited for each node in a *CG*, each packet from any source node to its destination node can first go upward to their least common ancestor and then go downward to the destination node. Therefore, there exists at least one path from one node to another. □

6 Experimental test

To evaluate the performance of the proposed routing algorithm, we implement four *Tree-turn* routing algorithms along with the *up*/down** routing, *L-turn* routing, and *up*/down** routing with DFS methodology on a simulator. In our network model, there are 8 ports in each switch, and each port is associated with one input channel and one output channel. We do not allow duplicated links between a pair of switches, that is, there exists at most one link between a pair of switches. The switching technique in switches is virtual cut-through. The packet length is 32 flits. We assume that the flying time of a packet between devices (endnode-to-switch and switch-to-switch) is 4 clock cycles. The routing time of a packet from one input port to one output port of the crossbar in a switch is 24 clock cycles, including forwarding table lookup, arbitration, and message startup time. The traffic pattern is uniform.

We have four configurations of networks: irregular networks with 64 nodes and 160 links, irregular networks with 64 nodes and 192 links, irregular networks with 128 nodes and 360 links, and irregular networks with 128 nodes and 400 links. For each kind of network configuration, we randomly generate 10 topologies and use the average performance of these 10 topologies as the performance of the configuration.

The simulation results are shown in Figs. 7, 8, 9 and 10. The throughput (accepted traffic) is defined as the received data per clock cycle per node (flits/cycle/node). The message latency is measured in clock cycles. “Tree-turn”, “BFS up*/down*”, “L-turn”, and “DFS up*/down*” denote *Tree-turn* routing, *up*/down** routing, *L-turn* routing, and *up*/down** routing with DFS methodology in the figures, respectively. Four network configurations, irregular networks with 64 nodes and 160 links, irregular networks with 64 nodes and 192 links, irregular networks with 128 nodes and 360 links, and irregular networks with 128 nodes and 400 links are shown in Figs. 7, 8, 9, and 10, respectively. We can see that *Tree-turn* routing outperforms other routing algorithms in these four network configurations. Because of the carefully selected prohibited turns in the proposed *Tree-turn* model, the derived *Tree-turn* routing is able to push the traffic downward in a spanning tree and we thus are able to get better performance.

Small networks (64 nodes) with different number of links (160 and 192) are shown in Figs. 7 and 8, respectively. We can see that *Tree-turn* routing outperforms the other algorithms. When comparing Fig. 7 and Fig. 8, the number of nodes in the network is the same, and we find that with more links, we can get more throughput.

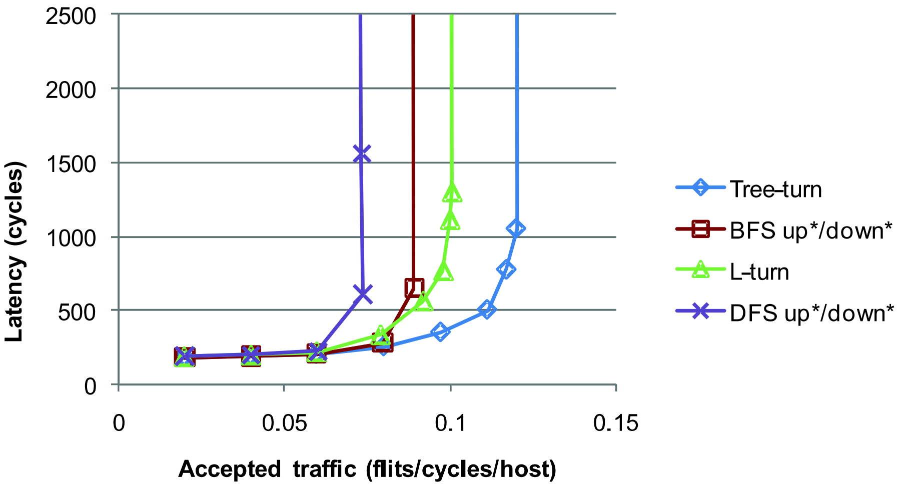


Fig. 7 Simulation result of the irregular networks with 64 nodes and 160 links

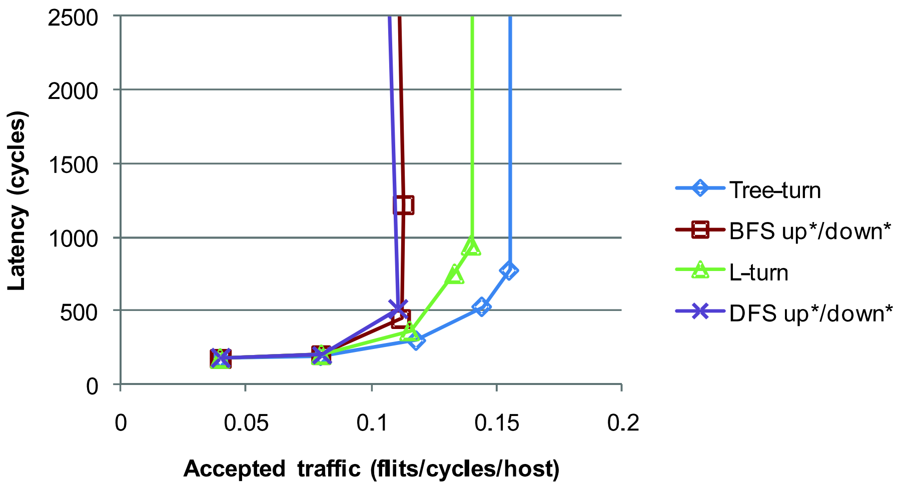


Fig. 8 Simulation result of the irregular networks with 64 nodes and 192 links

In Fig. 11, we compare the maximum throughput of each algorithm for the same network configuration. For the networks with 64 nodes and 160 links, the speedup of the throughput is 1.63 when comparing *Tree-turn* algorithm to *up*/down** routing with DFS methodology. For the networks with 64 nodes and 192 links, the speedup of the throughput is 1.40 when comparing *Tree-turn* algorithm to *up*/down** routing with DFS methodology.

Large networks (128 nodes) with different number of links (360 and 400) are shown in Figs. 9 and 10, respectively. We can see that *Tree-turn* routing outperforms the other algorithms. When comparing Fig. 9 and Fig. 10, the number of nodes in the

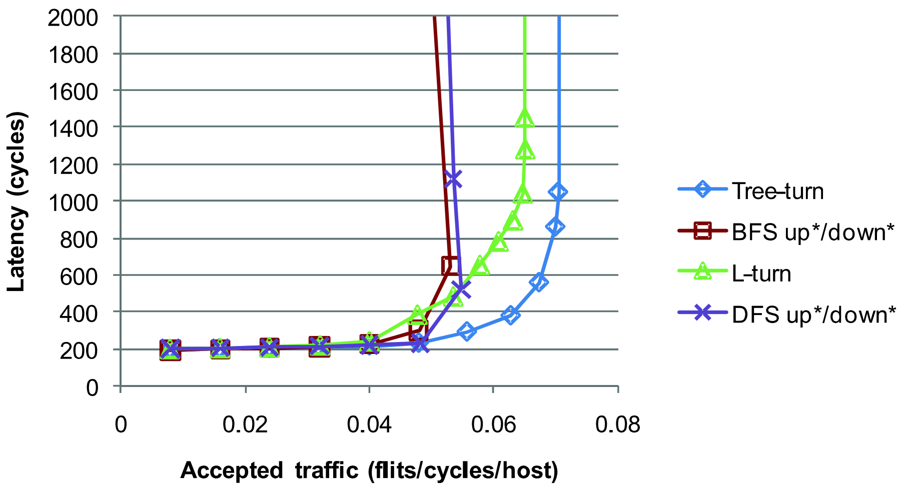


Fig. 9 Simulation result of the irregular networks with 128 nodes and 360 links

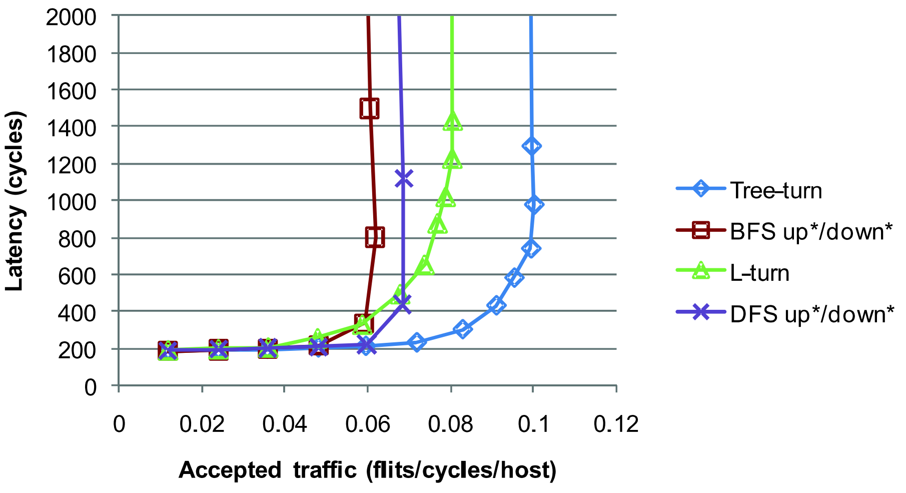


Fig. 10 Simulation result of the irregular networks with 128 nodes and 400 links

network is the same, and we find that with more links, we get more throughput. This observation is similar to that of small networks. In Fig. 12, we compare the maximum throughput of each algorithm for the same network configuration. For the networks with 128 nodes and 360 links, the speedup of the throughput is 1.33 when comparing *Tree-turn* algorithm to *up*/down** routing. For the networks with 128 nodes and 400 links, the speedup of the throughput is 1.62 when comparing *Tree-turn* algorithm to *up*/down** routing.

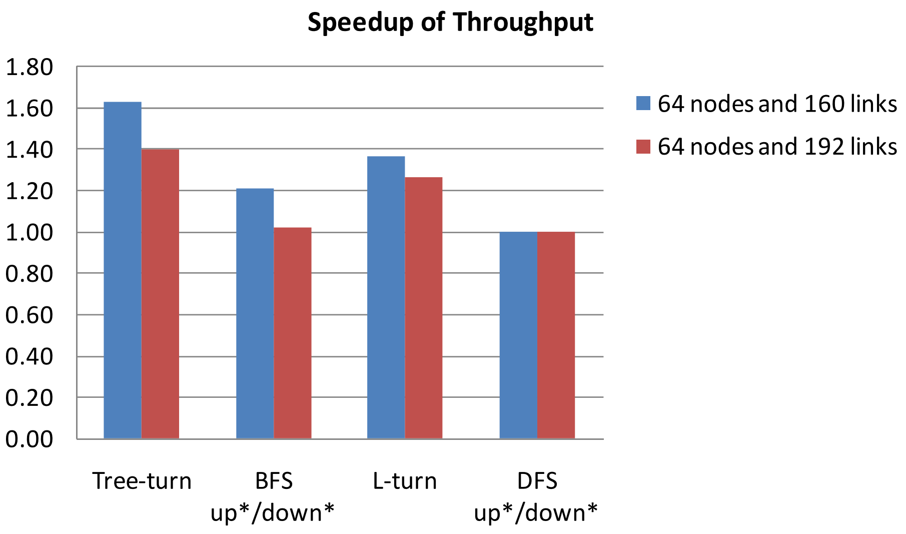


Fig. 11 Speedup of throughput for all algorithms compared to up*/down* algorithm with DFS methodology on the irregular networks with 64 nodes and 160 links and the irregular networks with 64 nodes and 192 links

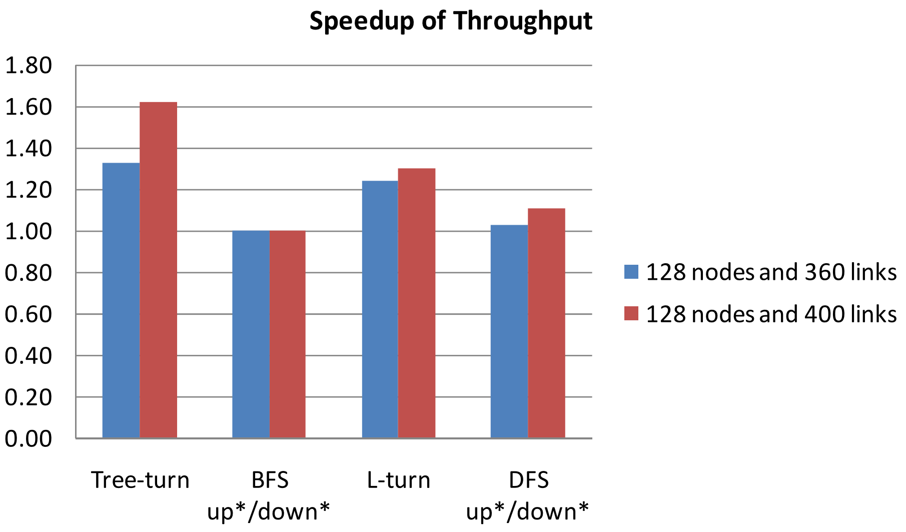


Fig. 12 Speedup of throughput for all algorithms compared to up*/down* algorithm on the irregular networks with 128 nodes and 360 links and the irregular networks with 128 nodes and 400 links

7 Conclusions and future work

In this paper, we propose a *Tree-turn* model for irregular topologies. The *Tree-turn* model is a tool used to develop a deadlock-free routing algorithm. Based on the *Tree-turn* model, we derive an efficient deadlock-free routing algorithm, which we call *Tree-turn* routing. To evaluate the performance of *Tree-turn* routing algorithm, we develop a simulator and implement the proposed *Tree*-turn routing along with *up*/down** routing, *L-turn* routing, and *up*/down** routing with DFS methodology. The simulation results show that the proposed *Tree-turn* routing outperforms other routing algorithms for all the test cases. We find that in the small networks with 64 nodes and 160 links, the speedup of throughput is 1.63 when we compare the *Tree-turn* routing to *up*/down** routing with DFS methodology. In the large networks with 128 nodes and 400 links, the speedup of throughput is 1.62 when we compare the *Tree-turn* routing to *up*/down** routing.

In the future, we will extend our work to regular topologies like 2-D mesh and 3-D torus networks. To provide efficient deadlock-free routing algorithms on these regular networks, we will need to choose different set of prohibited turns in the *Tree-turn* model with the consideration of characteristics of regular networks.

References

1. Boden NJ, Cohen D, Felderman RE, Kulawik AE, Seitz CL, Seizovic JN, Su W-K (1995) Myrinet: a gigabyte-per-second local area network. In: IEEE Micro, February 1995, pp 29–36
2. Cherkasova L, Kotov V, Rokicki T (1996) Fibre channel fabrics: evaluation and design. In: Proceedings of annual Hawaii international conference on system science, January 1996, pp 53–62
3. Chiu G-M (2000) The odd-even turn model for adaptive routing. IEEE Trans Parallel Distrib Syst 11(7):729–738
4. Dally WJ (1992) Virtual-channel flow control. IEEE Trans Parallel Distrib Syst 3(2):194–205
5. Dally WJ, Seitz CL (1987) Deadlock-free message routing in multiprocessor interconnection networks. IEEE Trans Comput 36(5):547–553
6. Dally WJ, Aoki H (1993) Deadlock-free adaptive routing in multicomputer networks using virtual channels. IEEE Trans Parallel Distrib Syst 4(4):466–475
7. Duato J (1995) A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. IEEE Trans Parallel Distrib Syst 6(10):1055–1067
8. Duato J (1994) A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In: Proceedings of 1994 IEEE international conference on parallel processing, August 1994, pp 142–149
9. Duato J (1993) A new theory of deadlock-free adaptive routing in wormhole networks. IEEE Trans Parallel Distrib Syst 4(12):1320–1331
10. Duato J (1991) On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies. In: Proceedings of parallel architectures and languages Europe 91, vol 1, June 1991, pp 390–405
11. Glass CJ, Ni LM (1992) Maximally fully adaptive routing in 2D meshes. In: Proceedings of IEEE international conference on parallel processing, vol 1, August 1992, pp 101–104
12. Glass CJ, Ni LM (1994) The turn model for adaptive routing. J ACM 5:874–902
13. Glass CJ, Ni LM (1992) The turn model for adaptive routing. In: Proceedings of the 19th international symposium on computer architecture, May 1992, pp 278–287
14. InfiniBand Trade Association (2004) InfiniBand architecture specification, vol 1, Release 1.2, October 2004. http://infinibandta.org/specs/
15. Jouraku A, Koibuchi M, Amano H, Funahashi A (2007) An effective design of deadlock-free routing algorithms based on 2D turn model for irregular networks. IEEE Trans Parallel Distrib Syst 18(3):320–333
16. Jouraku A, Koibuchi M, Amano H, Funahashi A (2002) Routing algorithms based on 2D turn model for irregular networks. In: Proceedings of the IEEE international symposium on parallel architectures, algorithms, and networks, May 2002, pp 254–259
17. Koibuchi M, Funahashi A, Jouraku A, Amano H (2001) L-turn routing: an adaptive routing in irregular networks. In: Proceedings of IEEE international conference on parallel processing, September 2001, pp 383–392

18. Koibuchi M, Jouraku A, Watanabe K, Amano H (2003) Descending layers routing: a deadlock-free deterministic routing using virtual channels in system area networks with irregular topologies. In: Proceedings of international conference on parallel processing, October 2003, pp 527–536
19. Lin X, McKinley PK, Ni LM (1995) The message flow model for routing in wormhole-routed networks. IEEE Trans Parallel Distrib Syst 6(7):755–760
20. Lin XY, Chung YC, Huang TY (2004) A multiple LID routing scheme for fat-tree-based InfiniBand networks. In: Proceedings of IEEE international parallel and distributed proceeding symposiums (CD-ROM), April 2004
21. Lysne O, Skeie T, Reinemo S-A, Theiss I (2006) Layered routing in irregular networks. IEEE Trans Parallel Distrib Syst 17(1):51–65
22. Mejia A, Flich J, Duato J, Reinemo S-A, Skeie T (2006) Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. In: Proceedings of international parallel and distributed processing symposium, April 2006
23. Ni LM, McKinley PK (1993) A survey of wormhole routing techniques in direct networks. IEEE Comput 26(2):62–67
24. Petrini F, Feng W-C, Hoisie A, Coll S, Frachtenberg E (2001) The quadrics network (QsNet): high-performance clustering technology. In: IEEE hot interconnects, August 2001, pp 125–130
25. Puente V, Gregorio JA, Beivide R, Vallejo F, Ibanez A (2001) A new routing mechanism for networks with irregular topology. In: Proceedings of the 2001 ACM/IEEE conference on supercomputing (CDROM), pp 1–8
26. Sancho JC, Robles A, Duato J (2000) A flexible routing scheme for networks of workstations. In: Proceedings of international conference on high performance computing, October 2000, pp 260–267
27. Sancho JC, Robles A, Duato J (2004) An effective methodology to improve the performance of the up*/down* routing algorithms. IEEE Trans Parallel Distrib Syst 15(8):740–754
28. Schroeder MD, Birrell AD, Burrows M, Murray H, Needham RM, Rodeheffer TL, Satterthwaite EH, Thacker CP (1990) Autonet: a high-speed, self-configuring local area network using point-to-point links. Technical Report SRC Research Report 59, DEC, April 1990
29. Schwiebert L, Jayasimha DN (1995) A universal proof technique for deadlock-free routing in interconnection networks. In: Proceedings of symposium on parallel algorithms and architectures, July 1995, pp 175–184
30. Silla F, Duato J (2000) High-performance routing in networks of workstations with irregular topology. IEEE Trans Parallel Distrib Syst 5(7):699–719
31. Silla F, Duato J (1997) Improving the efficiency of adaptive routing in networks with irregular topology. In: Proceedings of the 1997 conference on high performance computing, December 1997
32. Skeie T, Lysne O, Flich J, López P, Robles A, Duato J (2004) LASH-TOR: a generic transition-oriented routing algorithm. In: Proceedings of international conference on parallel and distributed systems, July 2004, pp 595–604
33. Wu J (2003) A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model. IEEE Trans Parallel Distrib Syst 52(9):1154–1169