A Tree-Turn Model for Irregular Networks

Jiazheng Zhou, Xuan-Yi Lin, and Yeh-Ching Chung[1]
Department of Computer Science, National Tsing Hua University,
Hsinchu, Taiwan 30013, R.O.C
Email: {jzzhou, xylin, ychung}@cs.nthu.edu.tw

Abstract

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

1. Introduction

Efficient routing algorithms can achieve high performance in switched-based networks such as Myrinet [1], ServerNet [5], and InfiniBand [6]. For wormhole switching [12], the deadlock is occurred when cyclic waiting of messages. How to design an efficient deadlock-free [2] routing algorithm is important to such networks. Many methods to prevent deadlocks have been proposed in the literature [3][10][11][15]. The turn model proposed in [4] was a tool to deliver deadlock-free routing algorithms for regular topologies. It analyzes the directions of messages and prohibits enough turns to break the turn cycles to avoid deadlocks.

The *up**/*down** routing [14] was the first tree-based routing algorithm for irregular topology. In *up**/*down** routing, there are only two directions, up and down, for channels. A legal route of *up**/*down** routing follows the rule: it must traverse zero or more links in the up direction followed by zero or more links in the down direction. Although the *up**/*down** routing is simple, the performance is not good since there exists many traffic congestions at root of a spanning tree called *hot spots* [13][16].

To overcome the drawbacks of the *up*/down* routing*, the *L-turn routing* was proposed in [9] based on the 2D turn model [8]. In *L-turn* routing, there are four directions, left-up, left-down, right-up, and right-down, for channels. The routing is based on the *L-R tree*. 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. 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 as the same type of links. It is possible that the *hot spots* will still occur around the root under some *L-R trees*. It is also possible that the opposite prohibited turn pairs exist on a node and make traffic load unbalancing.

In this paper, we first propose a general turn model, *Tree-turn* model, for irregular topologies. In the *Tree-turn* model, the directions of channels can be classified into left-up, left, left-down, right-up, right, and right-down directions. It has two more directions, left and right, than the 2D turn model. With two more directions, there are more choices of routing paths. In addition, tree links and cross links are associated with different definitions (directions). The tree links can only have left-up and right-down directions and the cross links have left, left-down, right-up, and right directions. By giving different definitions to tree links and cross links, we can use cross links to push the traffic downward in a spanning tree and release hot spots.

Based on the *Tree-turn* model, we propose an efficient tree-based routing algorithm, *Tree-turn* routing, for irregular topologies. The principle of *Tree-turn* routing is to provide more bandwidth and push the traffic

[1] The corresponding author

Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)
0-7695-2640-3/06 $20.00 © 2006 IEEE

downward to leaves to prevent hot spots. Given an irregular topology $G = (V, E)$, to construct the *Tree-turn* routing, we first build up the corresponding coordinated tree $CT = (V, E')$ of G followed by constructing the communication graph $CG = (V, \vec{E})$ from G and CT. Then we can obtain the forwarding tables of nodes by using the all-pairs shortest path algorithm according to the prohibited turns derived from the *Tree-turn* model and the directions of the channels in CG.

To evaluate the performance of *Tree-turn* routing, we compare it with *up*/down** routing and *L-turn* routing. We implement a wormhole routing simulator for these three routing algorithms. We use six network sizes as test cases to evaluate the three routing algorithms. The simulation results show that *Tree-turn* routing outperforms other two routing algorithms for all test cases.

The rest of the paper is organized as follows. Some definitions and terms used in this paper will be given in Section 2. In Section 3, we will describe the *Tree-turn* model. The *Tree-turn* routing derived from *Tree-turn* model will be given in Section 4. The experimental test will be given in Section 5. We give the conclusions in Section 6.

2. Preliminaries

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

Definition 1 (*Graph*): Given a switch-based network, it can be represented as a graph $G = (V, E)$, where V is the set of the switches and 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, it consists of two communication channels $<v_i, v_j>$ and $<v_j, v_i>$ such that node v_i can send message to node v_j through $<v_i, v_j>$ and node v_j can send message to node v_i through $<v_j, v_i>$. For channel $<v_i, v_j>$, v_i and v_j are called *start* and *sink* nodes of the channel, respectively. $<v_i, v_j>$ is called the *output* channel and *input* channel of v_i and v_j, respectively.

Definition 2 (*Coordinated tree*): Given $G = (V, E)$, a *coordinated tree* (CT) is a breath first search (BFS) 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.

Due to two or more children nodes can be selected as the next preorder traversal node, several coordinated trees can be built from the same network topology. To obtain the unique coordinated tree of a given network topology, the node with smaller network ID will be selected first 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 $\vec{e} = <v_i, v_j> \in \vec{E}$, 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 $\vec{e} = <v_i, v_j>$, the direction of $\vec{e}$, denoted as $d(\vec{e})$, is defined as *LU*, *L*, *LD*, *RD*, *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 $\vec{e}_\alpha = <v_i, v_j>$ and $\vec{e}_\beta = <v_j, v_k>$. 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 $\vec{T} = \{T_{d_i, d_j} \mid$ for all $d_i, d_j \in D$ 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*.

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*.

Lemma 1: Given a *CG* and a *DDG* of *CG*, if there is no cycle in the *DDG*, then it is impossible to have turn cycles in *CG* when the edges of *DDG* are the only available turns allowed at each node in *CG*.

Proof: We want to show that if there is a turn cycle in a *CG*, then there is a cycle in *DDG*. Assume that there is a turn cycle $TC = (T_{d(\vec{e}_1),d(\vec{e}_2)}, T_{d(\vec{e}_2),d(\vec{e}_3)}, \ldots, T_{d(\vec{e}_k),d(\vec{e}_{k+1})})$ in *CG*. The turn cycle *TC* can be simply represented as $TC'(T_{d_1,d_2}, T_{d_2,d_3}, \ldots, T_{d_k,d_1})$ in the corresponding *DDG*, where $d_1 = d(\vec{e}_1) = d(\vec{e}_{k+1})$, $d_2 = d(\vec{e}_2)$, $d_3 = d(\vec{e}_3)$, …, and $d_k = d(\vec{e}_k)$. $TC'(T_{d_1,d_2}, T_{d_2,d_3}, \ldots, T_{d_k,d_1})$ is a cycle in the *DDG*.

We now give an example to explain above definitions. In Figure 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 Figure 1(b), a BFS spanning tree of the network in Figure 1(a) is shown. The root in the BFS spanning tree is node v_1. The *coordinated tree* of *G* is shown in Figure 1(c). In Figure 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 since 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 Figure 1(a) and Figure 1(c). We use thick links and thin links in Figure 1(d) to represent tree links and cross links, respectively. We can find that the directions of tree links are either *LU* or *RD*, and the directions of cross links are *L*, *LD*, *RU*, or *R*. In Figure 1(d), the directions $d(<v_1, v_2>) = RD$, $d(<v_2, v_1>) = LU$, $d(<v_2, v_3>) = R$, $d(<v_3, v_2>) = L$, $d(<v_4, v_5>) = LD$, and $d(<v_5, v_4>) = RU$. $T_{d(<v_2,v_1>),d(<v_1,v_3>)} = T_{LU,RD}$ is a *turn* and $TC = \{T_{d(<v_2,v_1>),d(<v_1,v_3>)}, T_{d(<v_1,v_3>),d(<v_3,v_2>)}, T_{d(<v_3,v_2>),d(<v_2,v_1>)}\} = \{T_{LU,RD}, T_{RD,L}, T_{L,LU}\}$ is a *turn cycle*.

In Figure 1(e), the *direction graph DG* of Figure 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 Figure 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 Figure 1(e). It has two turns $T_{L,RD}$ and $T_{RD,L}$. If we only allow these two turns in Figure 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*.

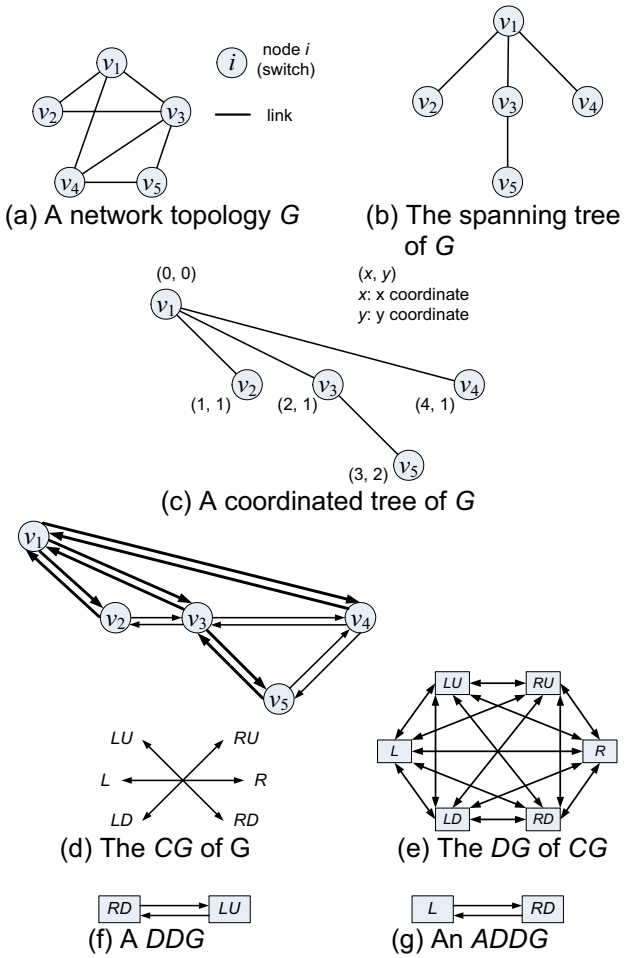


Figure 1. An example for definitions.

3. The *Tree-turn* Model

The *Tree-turn* model is a general turn model for irregular topology. Given an irregular topology *G*, in the *Tree-turn* model, based on Definitions 2, 3, 4, and 5, the directions of channels can be classified into six directions, 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. With these two

more directions, there are more choices of routing paths. 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, $d(\vec{e}) \in \{L, LD, RU, R, RD\}$. Tree links and cross links are associated with different directions in the *Tree-turn* model. By giving different directions to tree links and cross links, we can use cross links to push the traffic downward in a spanning tree and release hot spots.

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 apply the prohibited turns derived from the construction of a 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 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, when removing edges from a *DDG* in each step, we will try to prevent the traffic from flowing to the root of a *CG* and push the traffic downward to the leaves of a *CG*. 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 details.

A. 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 Figure 2, we must remove one edge from each *DG*. In Figure 2(a), we remove the edge $T_{RD,LU}$ and this is the only choice. The reason is to maintain the connectivity of a topology. Since the *LU* and *RD* directions are defined for tree links, if the topology is a tree and we remove edge $T_{LU,RD}$, there is no way for all nodes to communicate with each other if one node is not the ancestor or the child of the other node. By removing edge $T_{RD,LU}$ from Figure 2(a), we can get $ADDG_1$ shown in Figure 3(a). In Figure 2(b), we can break the cycle by removing either edge of the *DG*. For each node *v* in the *CG*, the direction *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 to traffic downward, we keep edge $T_{LD,RU}$. By removing edge $T_{RU,LD}$ from Figure 2(b), we can get $ADDG_2$ shown in Figure 3(b). In Figure 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 figure 2(c), we can get $ADDG_3$ shown in Figure 3(c).

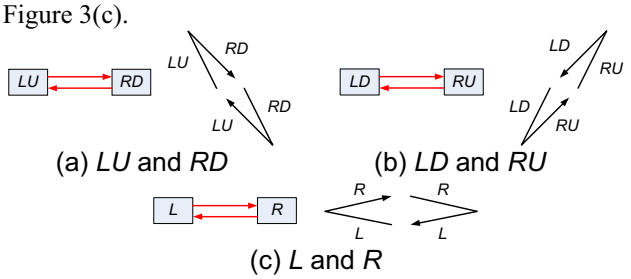
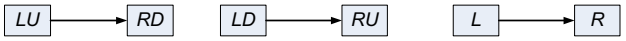


Figure 2. The *DG*s of node pairs and their corresponding possible turn cycles.



(a) $ADDG_1$ (b) $ADDG_2$ (c) $ADDG_3$

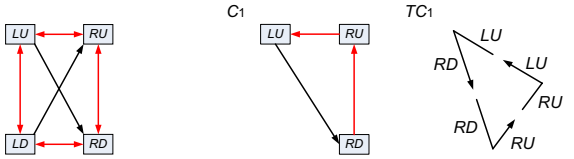
Figure 3. The maximal *ADDG*s of *DG*s shown in Figure 2.

B. 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 Figure 4(a). In Figure 4(a), there are four cycles C_1, C_2, C_3, and C_4 that will result in turn cycles TC_1

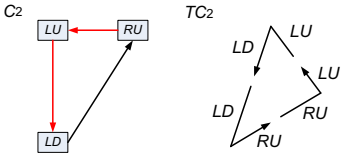
$= \{ T_{RD,RU}, T_{RU,LU}, T_{LU,RD} \}$, $TC_2 = \{ T_{LD,RU}, T_{RU,LU}, T_{LU,LD} \}$, $TC_3 = \{ T_{RU,RD}, T_{RD,LD}, T_{LD,RU} \}$, and $TC_4 = \{ T_{RD,LD}, T_{LD,LU}, T_{LU,RD} \}$ in a *CG* as shown in Figures 4(b), 4(c), 4(d), and 4(e), respectively. To break these four turn cycles, we need to remove some edges from the *DDG* shown in Figure 4(a).

For cycles C_1 and C_2, they have a common edge $T_{RU,LU}$ and this edge makes the traffic flow upward. In order to push the traffic flow downward to 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_3, since *LU* and *RD* are directions of tree links, we cannot remove $T_{LU,RD}$ for connectivity reason as stated in Step 1. Therefore, we remove edge $T_{LD,LU}$ to break cycle C_4. We then obtain the $ADDG_4$ as shown in Figure 4(f).

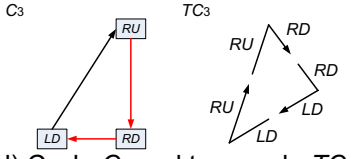


(a) The *DDG* by combining $ADDG_1$ with $ADDG_2$

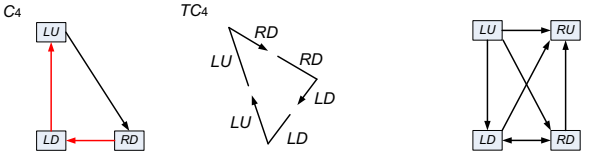
(b) Cycle C_1 and turn cycle TC_1



(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) $ADDG_4$

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

C. 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 new formed *DDG*. For nodes in Figure 4(f), we have the following observations:

Observation 1: Any combination of edges from nodes *LD* and *RD* would not have upward directions in a *CG*.

Observation 2: Any combination of edges from nodes *LU* and *RU* would not have downward directions in a *CG*.

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

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

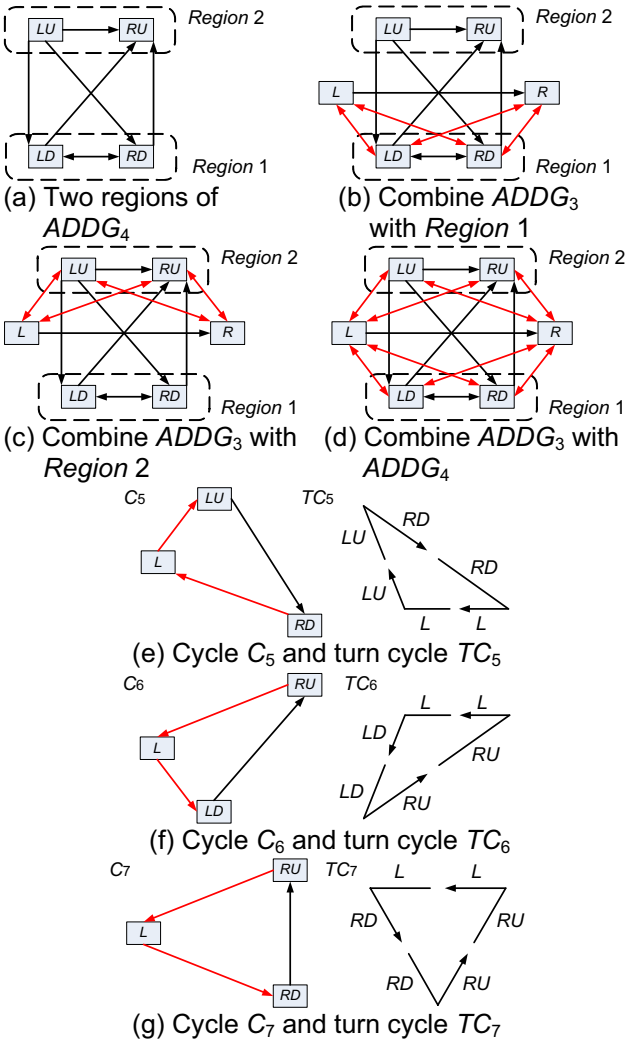
Observation 5: If we combine $ADDG_3$ with $ADDG_4$, there are two possible ways to form turn cycles. One is from node *v* in $ADDG_3$ to nodes in *Region* 1, nodes in *Region* 2, and goes back to node *v*. The other is from node *v* in $ADDG_3$ to nodes in *Region* 2, nodes in *Region* 1, and goes back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

Based on observations 3-5, in Figure 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 Figure 5(e), Figure 5(f), Figure 5(g), and Figure 5(h), Figure 5(i), and Figure 5(j), 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 connectivity reason. In order to push the traffic flow downward to leaves of a corresponding *CT*, we remove edge $T_{L,LU}$ to break cycle C_5. For cycles C_6 and C_7, they have a common edge $T_{RU,L}$ that makes the traffic flow upward. In order to push the traffic flow downward to leaves of a corresponding *CT*, we remove edge $T_{RU,L}$ to break cycles C_6, and C_7. For cycle C_8, since only edge $T_{RU,R}$

makes the traffic flow upward ($T_{LD,RU}$ makes the traffic flow downward then upward), in order to push the traffic flow downward to leaves of a corresponding *CT*, we remove edges $T_{RU,R}$ to break cycle C_8. For cycles C_9 and C_{10}, they have a common edge $T_{R,LU}$ that makes the traffic flow upward. In order to push the traffic flow downward to leaves of a corresponding *CT*, we remove edge $T_{R,LU}$ to break cycles C_9, and C_{10}. We obtain $ADDG_5$ as shown in Figure 5(k).

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 the *Tree-turn* model.



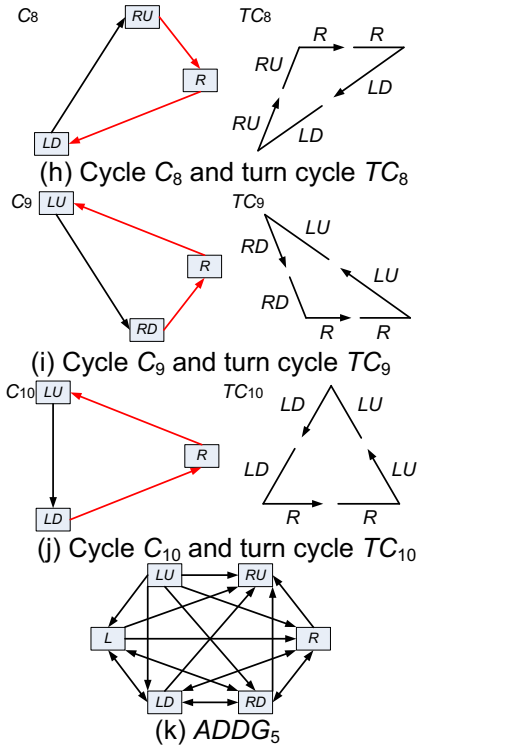


Figure 5. Combine $ADDG_3$ with $ADDG_4$ to form $ADDG_5$.

4. The *Tree-turn* Routing

Based on the *Tree-turn* model, given an irregular topology $G = (V, E)$, we can derive the *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 the *Tree-turn* model and the directions of the channels in *CG*.

In phase 3, for the all-pairs shortest path algorithm, whenever we find a shorter routing path through node *k*, and if the turn formed at node *k* is not a prohibited turn, we will adjust the routing path and setup the forwarding tables of the nodes on the routing path. Otherwise, we will keep the original routing path. If there are several routing paths with the same length, we add all of them to the routing tables. Following is the detail of all-pairs shortest path algorithm for *Tree-turn* routing.

Algorithm setup_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.
2. Let *length*[i][j] be the length from node i to node j.
3. Let *direction*(i, j) be the direction of channel <i, j> formed by node i and node j.
4. Let *turn*(d_i, d_j) be the turn form direction d_i and direction d_j.
5. /* Initialize the *length*[i][j] according to the adjacency matrix. */
for i = 1 to n **do**
  for j = 1 to n **do**
    if (there exists one link between node i and node j)
    then { *length*[i][j] = 1; }
6. /* 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 (*length*[i][k] + *length*[k][j] <=*length*[i][j]) **then**
      {
        Let node m be the (*length*[i][k] – 1)th node of *routing_path*[i][k].
        Let node n be the second node of *routing_path*[k][j].
        inDirection = *direction*(m, k);
        outDirection = *direction*(k, n);
        if (*turn*(*inDirection*, *outDirection*) is not prohibited) **then**
        {
          length[i][j] = *length*[i][k] + *length*[k][j];
          Adjust the routing paths and setup the forwarding tables for the nodes on the routing paths.
        }
      }
End_of_algorithm_setup_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 *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 goes downward to the destination node. Therefore, there exists at least one path from one node to another.

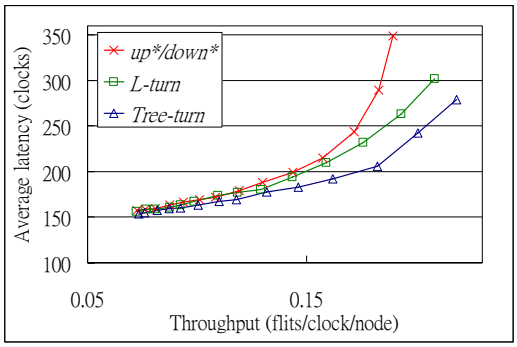
5. Simulation Results

To evaluate the performance of the proposed routing algorithm, we implement the *Tree-turn* routing algorithm along with the *up*/down** routing algorithm and the *L-turn* routing algorithm on a wormhole routing simulator, IRFlexSim [7]. In our network model, the topologies are generated randomly by given number of switches and links. Each switch is associated with a processor. 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 packet length is 128 flits. The delay for a flit goes through a link is one clock. The delay for the flit header to be routed and arbitrated to the output channel is one clock. The delay for a data flit to be transmitted from the input channel to the output channel is one clock. The traffic pattern is uniform. To simulate the irregular topology, we have six configurations for different number of nodes (switches) n and links l, that is, (n, l) ∈ {(128, 384), (128, 448), (128, 512)}.

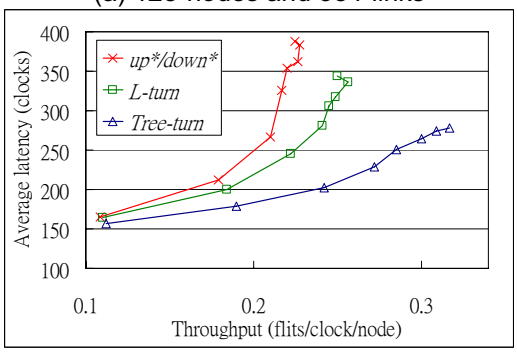
Figure 6 shows the simulation results of these three algorithms under different network configurations. In Figure 6, the throughput is defined as the received data per clock per node (flits/clock/node). The message latency is measured in clocks. From the simulation results, we can see that the performance of *Tree-turn* routing is better than that of *L-turn* routing, and the performance of *L-turn* routing is better than that of *up*/down** routing. That is, *Tree-turn* routing outperforms *L-turn* routing and *up*/down** routing. For topologies used in Figures 6(a) to 6(c), they have the same number of nodes, but different numbers of links. From Figures 6(a) to 6(c), for all routing algorithms, we can see that when the number of links increases, the throughputs of routing algorithms are getting larger and the latencies of routing algorithms are getting smaller.

6. Conclusions

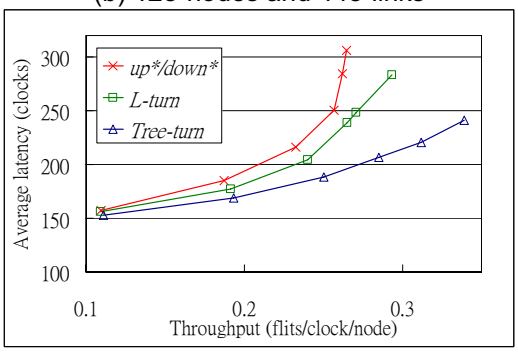
In this paper, we have proposed a general *Tree-turn* model for irregular topology. Based on the *Tree-turn* model, we derive an efficient deadlock-free routing algorithm, *Tree-turn* routing. To evaluate the performance of the proposed routing algorithm, we have implemented the *Tree-turn* routing algorithm along with the *up*/down** routing algorithm and the *L-turn* routing algorithm on a software simulator. The simulation results show that the proposed *Tree-turn* routing outperforms other two routing algorithms for all the test cases.



(a) 128-nodes and 384-links



(b) 128-nodes and 448-links



(c) 128-nodes and 512-links

Figure 6. Simulation results of different routing algorithms with different network configurations.

Acknowledgement

The work of this paper is partially supported by National Science Council, Ministry of Economic Affairs of the Republic of China under contract NSC-94-2213-E-007-080, NSC-95-2752-E-007-004-PAE and MOEA-95-EC-17-A-04-S1-044.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, pp. 29-36, February 1995.

[2] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547-553, May 1987.

[3] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055-1067, October, 1995.

[4] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," *Journal of ACM*, vol. 5, pp. 874-902, September 1994.

[5] R. Horst, "ServerNet Deadlock Avoidance and Fractahedral Topologies," *Proceedings of 10th International Parallel Processing Symposium*, pp. 274-280, April 1996.

[6] InfiniBand Trade Association, "InfiniBand Architecture Specification, Volume 1, Release 1.2," October 2004. http://infinibandta.org/specs/.

[7] IRFlexSim 0.5. http://ceng.usc.edu/smart/tool.htm.

[8] A. Jouraku, M. Koibuchi, H. Amano, and A. Funahashi, "Routing Algorithms Based on 2D Turn Model for Irregular Networks." *Proceedings of the IEEE International Symposium on Parallel Architectures, Algorithms, and Networks*, pp. 254-259, May 2002.

[9] M. Koibuchi, A. Funahashi, A. Jouraku, and H. Amano, "L-turn Routing: An Adaptive Routing in Irregular Networks," *Proceedings of IEEE International Conference on Parallel Processing*, pp. 383-392, September 2001.

[10] X. Lin, P. K. McKinley, and L. M. Ni, "The Message Flow Model for Routing in Wormhole-Routed Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 7, pp. 755-760, July 1995.

[11] X. Lin, P. K. McKinley, and L. M. Ni, "The Message Flow Model for Routing in Wormhole-Routed Networks," *Proceedings of 1993 IEEE International Conference on Parallel Processing*, pp. 294-297, August 1993.

[12] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, vol. 26, no. 2, pp. 62-67, February 1993.

[13] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control," *High Performance Interconnects for Distributed Computing*, July 2005.

[14] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker, "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," Technical Report SRC Research Report 59, DEC, April 1990.

[15] L. Schwiebert and D. N. Jayasimha, "A Universal Proof Technique for Deadlock-Free Routing in Interconnection Networks," *Proceedings of Symposium on Parallel Algorithms and Architectures*, pp. 175-184, July 1995.

[16] X. Zhang, Y. Yan, and R. Castaneda, "Comparative Performance Evaluation of Hot Spot Contention Between MIN-based and Ring-based Shared-Memory Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 8, pp. 872-886, August 1995.