International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 1, Number 2, pages 178–190, July 2011

Node-to-Set Disjoint-Paths Routing in Recursive Dual-Net

Yamin Li
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan

Shietung Peng
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan

Wanming Chu
Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan

Received: January 28, 2011
Revised: April 28, 2011
Accepted: June 20, 2011
Communicated by Sayaka Kamei

Abstract

Recursive dual-net (RDN) is a newly proposed interconnection network for massive parallel computers. The RDN is based on recursive dual-construction of a symmetric base-network B. A k-level dual-construction for $k > 0$ creates a network $RDN^k(B)$ containing $N = (2n_0)^{2^k}/2$ nodes with node-degree $d_0 + k$, where n_0 and d_0 are the number of nodes and the node-degree of the base network, respectively. The RDN is a symmetric graph and can contain huge number of nodes with small node-degree and short diameter. Node-to-set disjoint-paths routing is fundamental and has many applications for fault-tolerant and secure communications in a network. In this paper, we propose an efficient algorithm for node-to-set disjoint-paths routing in RDN. We show that, given a node s and a set of $d_0 + k$ nodes T in $RDN^k(B)$, $d_0 + k$ disjoint paths, each connecting s to a node in T, can be found in $O(((d_0 + k)D_0/\lg n_0)\lg N)$ time, and the length of the paths is at most $3(D_0/2+1)(\lg N+1)/(\lg n_0+1)$, where N is the number of nodes in $RDN^k(B)$, d_0 , D_0, and n_0 are the node-degree, the diameter, and the number of nodes of base-network B, respectively.

1 Introduction

The demand for more computing power has never stopped. The performance of processors has doubled in every 18-month. Parallel computer systems with large number of processors achieved petaflops computing performance and are opening the door to exaflops. In the last decade, because

of the advance in computer technology, computer makers such as IBM and Cray have risen up competition to build supercomputers with hundreds of thousands of processors. It has been predicted that, in the near future, the number of processors will reach millions [3].

Interconnection networks play a critical role for those supercomputers to gain high-performance. It is possible to combine cheap and efficient products to provide almost all components of a parallel computer except for the interconnection network [7]. Therefore, many topologies have been proposed for the interconnection networks and studied eagerly [2, 5, 14, 15].

The recursive dual-net (RDN), a newly proposed interconnection network, is based on recursive dual-construction of a symmetric base-network [17]. The dual-construction extends a network with n nodes and node-degree d to a network with $2n^2$ nodes and node-degree $d+1$. The k-level RDN is obtained by recursively applying dual-construction k times starting from the symmetric base network B. The RDN has many merits as long as topological properties are concerned. For example, an RDN can connect a huge number of nodes with just a small number of links per node. It is not difficult to construct an RDN connecting 3-millions nodes with 6 links per node and its diameter equals 22. Therefore, it is an interesting candidate as an interconnection network for massively parallel computers of next generations.

In the research on interconnection networks, it is very important to design and develop efficient routing algorithms. The algorithms include those for solving disjoint-paths problems in node-to-node, node-to-set, and set-to-set routings. These problems are fundamental and essential for fault tolerance and security in parallel computation and communication.

The problem of disjoint-paths routing has been investigated in many topologies. There are several algorithms in star graphs for the node-to-node disjoint paths problem in $O(n^2)$ time [6, 11, 13]. Gu and Peng gave algorithms for the node-to-set and the set-to-set disjoint-paths problems in hypercubes in $O(n^2)$ and $O(n^2 \log n)$ time, respectively [10]. Gu and Peng gave algorithms for the node-to-set and the set-to-set disjoint-paths problems in star graphs in $O(n^2)$ time [8, 9]. In pancake graphs, there are algorithms for the node-to-node and the node-to-set disjoint-paths problems in $O(n^2)$ time [12, 18]. Bossard, Kaneko and Peng gave an algorithm for node-to-set disjoint-paths problem in metacube [4]. Li, Peng and Chu gave algorithms for node-to-node disjoint-paths and fault-tolerant routing problem in RDN [16].

In this paper, we propose an efficient algorithm that finds disjoint paths for node-to-set routing in recursive dual-nets. For a k-level recursive dual-net with base-network B, $RDN^k(B)$, the algorithm can find $k+d_0$ disjoint paths in $O(((d_0+k)D_0/\lg n_0)\lg N)$ time and the length of the paths is at most $3(D_0/2+1)(\lg N+1)/(\lg n_0+1)$, where N is the number of nodes in $RDN^k(B)$, d_0, D_0 and n_0 are the node-degree, the diameter, and the number of nodes of the base-network, respectively.

The rest of the paper is organized as follows. Section 2 introduces some terminologies, definitions, and basic properties of recursive dual-nets. Section 3 gives the proposed algorithms for node-to-set disjoint-paths problem in RDN. Finally, Section 4 concludes this paper with some possible future works.

2 Recursive Dual-Net

Let G be an undirected graph. The size of G, denoted as $|G|$, is the number of vertices. A path from node s to node t in G is denoted by $s \to t$. The length of the path is the number of edges in the path. For any two nodes s and t in G, we denote $D(s,t)$ as the length of a shortest path connecting s and t. The diameter of G is defined as $D(G) = \max\{D(s,t)|s,t \in G\}$. For any two nodes s and t in G, if there is a path connecting s and t, we say G is connected. If every node in G looks alike, we say G is symmetric.

Given a symmetric connected graph B with n_0 nodes and the node degree d_0, a Recursive Dual-Net of level k, denoted as $RDN^k(B)$ or $RDN^k(B(n_0))$, can be recursively defined as follows:

1. $RDN^0(B) = B$ is a symmetric connected graph with n_0 nodes, called *base network*;
2. For $k > 0$, an $RDN^k(B)$ is constructed from $RDN^{k-1}(B)$ by a dual-construction as explained below (also see Figure 1).

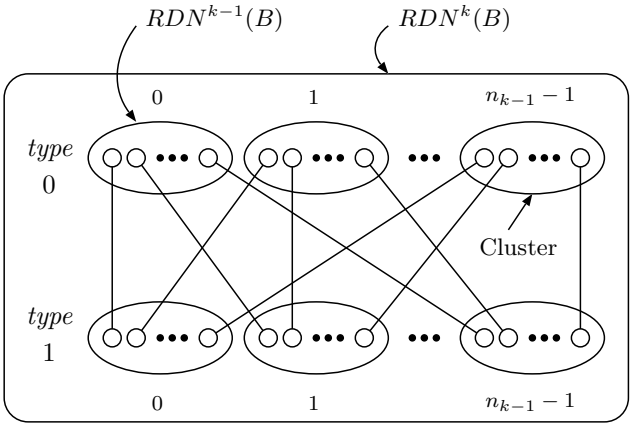

Figure 1: Build an $RDN^k(B)$ from $RDN^{k-1}(B)$

Dual-construction: Let $RDN^{k-1}(B)$ be referred to as a *cluster* of level k and the number of nodes $n_{k-1} = |RDN^{k-1}(B)|$. An $RDN^k(B)$ is a graph that contains $2n_{k-1}$ clusters of level k as subgraphs. These clusters are divided into two sets with each set containing n_{k-1} clusters. Each cluster in one set is said to be of *type* 0, denoted as C_i^0, where $0 \leq i \leq n_{k-1} - 1$ is the cluster ID. Each cluster in the other set is of *type* 1, denoted as C_j^1, where $0 \leq j \leq n_{k-1} - 1$ is the cluster ID. In the dual-construction at level k, each node has a new link to a node in a distinct cluster of the other type. We call this link *cross-edge* of level k. That is, for each pair of clusters C_i^0 and C_j^1, there is a unique edge connecting a node in C_i^0 and a node in C_j^1, $0 \leq i, j \leq n_{k-1} - 1$. In Figure 1, there are n_{k-1} nodes within each cluster $RDN^{k-1}(B)$.

We give two simple examples of recursive dual-nets with $k = 1$ and 2, in which the base-network is a ring with 3 nodes, in Figure 2 and Figure 3, respectively. Figure 2 depicts an $RDN^1(B(3))$ network. There are 3 nodes in the base network. Therefore, the number of nodes in $RDN^1(B(3))$ is 2×3^2, or 18. Figure 3 shows the $RDN^2(B(3))$ constructed from the $RDN^1(B(3))$ in Figure 2. We did not show all the nodes in the figure. The number of nodes in $RDN^2(B(3))$ is 2×18^2, or 648.

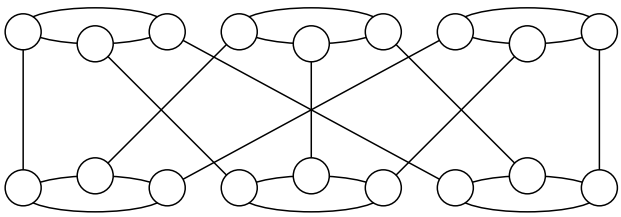

Figure 2: A Recursive Dual-Net $RDN^1(B(3))$

Similarly, we can construct an $RDN^3(B(3))$ containing 2×648^2, or 839,808 nodes with node-degree 5 and the diameter equals to 22. In contrast, the 839808-node 3D torus machine (adopt by IBM Blue Gene/L [1]) can be configured as $108 \times 108 \times 72$ nodes with node-degree 6 and the diameter of $54 + 54 + 36 = 144$.

A node presentation for $RDN^k(B)$ that provides a unique ID to each node in $RDN^k(B)$ is described as follows. Let the set of IDs of nodes in B, denoted as ID_0, be i, $0 \leq i \leq n_0 - 1$. The ID_k of node u in $RDN^k(B)$ for $k > 0$ is a triple (u_0, u_1, u_2), where u_0 is a 0 or 1, u_1 and u_2 belong to ID_{k-1}. We call u_0, u_1, and u_2 typeID, clusterID, and nodeID of u, respectively.

More specifically, ID_i, $1 \leq i \leq k$, can be defined recursively as follows: $ID_i = (b, ID_{i-1}, ID_{i-1})$, where $b = 0$ or 1, and ID_0 is the set of IDs of nodes in B. With this ID presentation, (u, v) is a cross-edge of level k in $RDN^k(B)$ iff $u_0 \neq v_0$, $u_1 = v_2$, and $u_2 = v_1$. The ID of a node u in $RDN^k(B)$ can also be presented by an unique integer i, $0 \leq i \leq (2n_0)^{2^k}/2 - 1$, where i is the

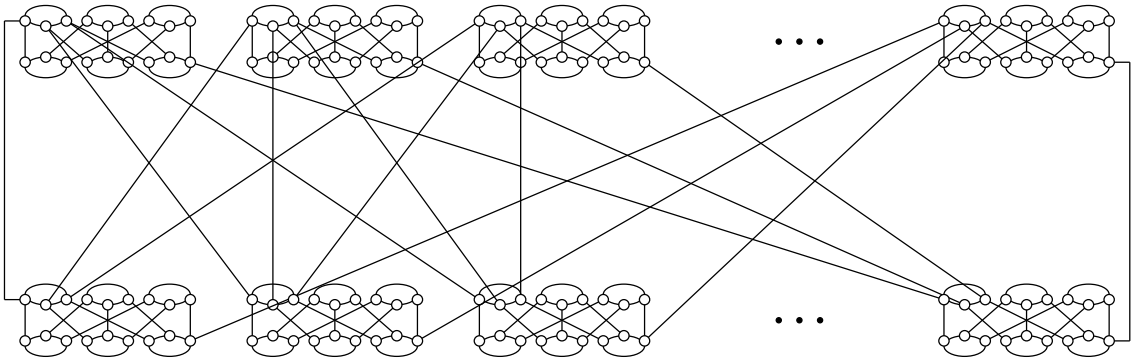

Figure 3: A Recursive Dual-Net $RDN^2(B(3))$

lexicographical order of the triple (u_0, u_1, u_2). For example, the ID of node $(1, 1, 2)$ in $RDN^1(B)$ is $1 * 3^2 + 1 * 3 + 2 = 14$.

The basic topological properties has been explored in [17]. The following lemma is from [17].

Lemma 1. *Assume that the base-network B is a symmetric graph with size n_0, node-degree d_0, and diameter D_0. Then, the size, the node-degree, the diameter, and the bisection bandwidth of $RDN^k(B)$ are $(2n_0)^{2^k}/2$, $d_0 + k$, $2^k D_0 + 2^{k+1} - 2$, and $\lceil (2n_0)^{2^k}/8 \rceil$, respectively.*

Table 1 shows the number of nodes an RDN can have when a hypercube or a 3D torus is used as the base network. The number in parentheses is the node-degree. We can see that an RDN can contain several hundreds to more than 1 billion nodes with less than or equal to 8 links per node.

Table 1: The number of nodes in RDN

Base network	$k = 0$	$k = 1$	$k = 2$
3-cube	8 (3)	128 (4)	32,768 (5)
4-cube	16 (4)	512 (5)	524,288 (6)
5-cube	32 (5)	2,048 (6)	8,388,608 (7)
3^3-torus	27 (6)	1,458 (7)	4,251,528 (8)
4^3-torus	64 (6)	8,192 (7)	134,217,728 (8)
5^3-torus	125 (6)	31,250 (7)	1,953,125,000 (8)

The number of nodes increases rapidly as the k increases. Maybe we need to find a way to control the increasing speed in the number of nodes that will result in a network rather than the pure RDN.

3 Node-to-Set Disjoint-Path Routing in RDN

In a graph G, given a source node s and a set of n destination nodes $T = \{t_1, \ldots, t_n\}$, the node-to-set disjoint-paths problem is to find n disjoint paths (sharing the common node s only), each connecting node s to a node in T. In this section, we will propose an efficient algorithm for the node-to-set disjoint-paths problem in RDN.

Given two nodes u and v in $RDN^k(B)$, there exists a simple routing algorithm that finds a shortest path from u to v [17]. The routing algorithm is described formally as Algorithm 1.

Lemma 2. *Assume that D_0 is the diameter of the base network B. Then, in $RDN^k(B)$ with $k > 0$, a path from source s to destination t can be found in $O(((D_0/\lg n_0) \lg N)$ time and the length of the path is at most $(D_0 + 2) * (\lg N + 1)/(\lg n_0 + 1) - 2$.*

Algorithm 1: RDN_routing($RDN^k(B), u, v$)
begin
  if $k = 0$ **then** RDN_routing(B, u, v)
  else
    Case 1: $u_0 = v_0$ and $u_1 = v_1$
      RDN_routing($RDN_u^{k-1}(B), u_2, v_2$);
      /* $RDN_u^{k-1}(B)$ is the cluster of level k, where node$u$ belongs to. */
    Case 2: $u_0 \neq v_0$
      RDN_routing($RDN_u^{k-1}(B), u_2, v_1$);
      $u' \leftarrow (u_0, u_1, v_1)$;
      RDN_routing($RDN_v^{k-1}(B), v_2, u_1$);
      $v' \leftarrow (v_0, v_1, u_1)$;
      connect u' and v' via a cross-edge of level k;
    Case 3: $u_0 = v_0$ and $u_1 \neq v_1$
      route u to w via the cross-edge of level k;
      route node w to node v as in Case 2;
  endif
end

Proof. Assume that RDN_routing(B, u, v) can find the shortest path connecting u and v in B in $O(D_0)$ time with the length of the path at most D_0. We show the correctness of Algorithm 1 by induction on k. By induction hypothesis, RDN_routing($RDN^{k-1}(B), u, v$) can find the shortest path connecting u and v in $RDN^{k-1}(B)$ in $O(D_{k-1})$ time with the length of the path at most D_{k-1}. From Algorithm 1, the worst-case occurs at Case 3, where u and v are in the distinct clusters of the same type. Since there is no direct connection between the two clusters, we have to route one of the two nodes to the cluster of the other type via a cross-edge of level k, and then follow the routing in Case 2. The routing for Case 2 first finds the unique cross-edge that connects the two clusters, and its two end-nodes u' and v'. Then, the routing can be done by two recursive calls for routing inside the two clusters ($u \rightarrow u'$ and $v \rightarrow v'$) and via the cross-edge of level k. Since the paths $u \rightarrow u'$ and $v \rightarrow v'$ are the shortest paths in $RDN^{k-1}(B)$, it is clear that the routing in Algorithm 1 is the shortest one. The length of the path is at most $2D_{k-1} + 2$ in the worst-case. From Lemma 1, $D_{k-1} = 2^{k-1}D_0 + 2^k - 2$ and $2^k = (\lg N + 1)/(\lg n_0 + 1)$. Therefore, the path can be found in $O(D_0 * 2^k) = O((D_0/\lg n_0) * \lg N)$ time, and the length of the path is at most $(2^k D_0 + 2^{k+1} - 4) + 2 = (D_0 + 2)2^k - 2 = (\lg N + 1)/(\lg n_0 + 1) - 2$. □

Let the $d_0 + k$ neighbors of node u be $u^{(i)}, 1 \leq i \leq d_0 + k$, where $u^{(i)}, 1 \leq i \leq d_0$, are the neighbor of s in B, and edge $(u, u^{(i)}), d_0 + 1 \leq i \leq d_0 + k$, is the cross-edge of level $i - d_0$. Let $u^{(i,j)} = (u^{(i)})^{(j)}$ for $1 \leq i, j \leq d_0 + k$, and so on. For simplicity, we denote $N(s) = \{s^{(i)}, 1 \leq i \leq d_0 + k\}$. C denotes a cluster (of level k). C_s denotes the cluster C with node $s \in C$. *type*(C) denotes the type of cluster C.

We give the following lemma which will be used frequently in our algorithms.

Lemma 3. *In $RDN^k(B)$, for any node u, there exist $d_0 + k$ disjoint paths $u \rightarrow u_i, 1 \leq i \leq d_0 + k$, of length at most 2 such that $u_i \notin C_u$ and $C_{u_i} \neq C_{u_j}$ if $i \neq j$.*

Proof. Let the $d_0 + k$ paths be $u \rightarrow u^{(i)} \rightarrow u^{(i,d_0+k)}, 1 \leq i \leq d_0 + k$, and $u \rightarrow u^{(d_0+k)}$. It is easy to see that these paths are disjoint and satisfy the conditions specified in the lemma. □

The strategy of the proposed algorithm to find disjoint paths is to connect the nodes in T but not in cluster C_s to node s via distinct clusters. In order to implement this strategy, we need to handle the clusters that contain more than one node in T. We should allow only one of them to be connected in the cluster and route all others out of the clusters by disjoint paths of length as small as possible. Once the nodes in T are routed into distinct clusters, if they are in the cluster of *type*(C_s), each of them can connect with one of nodes in $N(s)$, by RDN_routing. Otherwise, the

connection should go through an intermediate cluster. In such case, there are some conditions for selecting the intermediate cluster. The solutions for the problems described above are shown in the next paragraph.

To describe the algorithm that we propose for the node-to-set disjoint-paths routing in RDN, we need the following subroutines, namely, RDN_N2S_one_path and RDN_modified_routing. The subroutine RDN_one_path has three input parameters: node u and sets of nodes V and W in $RDN^k(B)$ that are defined in the main algorithm (Algorithm 4). It will be called when $|C_u \cap V| > 1$ to connect u to a node in $C_u \cap V$, and distribute all other nodes in $C_u \cap V$ to other distinct clusters not containing any node in W via disjoint paths of length at most 2 (see figure 4). In figure 4, $C_u \cap V = \{v_1, v_2, v_3\}$, u is connected to v_3 while v_1 and v_2 are routed out of cluster C_u via disjoint paths of length 2 and 1, respectively. It is assumed that $C_{v_1^{(d_0+k)}} \cap W \neq \emptyset$. Therefore, we need a path of length 2 for distributing v_1 to a node in another cluster.

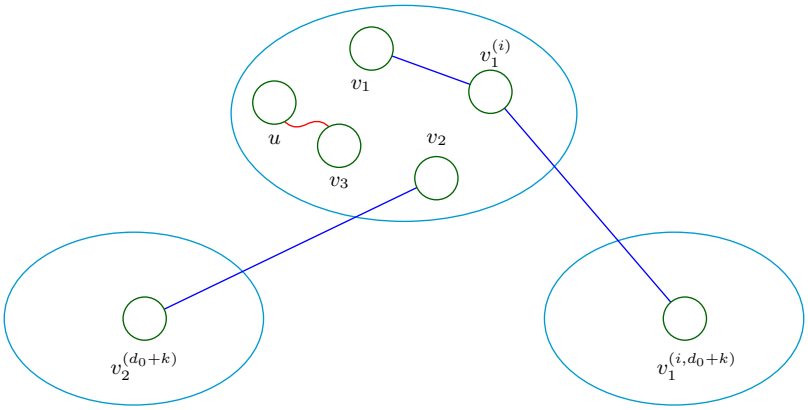

Figure 4: RDN_N2S_one_path

The subroutine RDN_modified_routing has three input parameters in $RDN^k(B)$: nodes u and v, and a cluster C with $C_u \neq C_v$, $type(C_u) = type(C_v)$, and $type(C) \neq type(C_u)$. It will be called to connect nodes u and v such that the path passes through clusters C_u, C, and C_v only (see figure 5). The two subroutines are described formally as Algorithms 2 and 3. The correctness of Algorithms 2 and 3, the upper bounds of time complexities and the maximum lengths of the paths are shown in Lemmas 4 and 5.

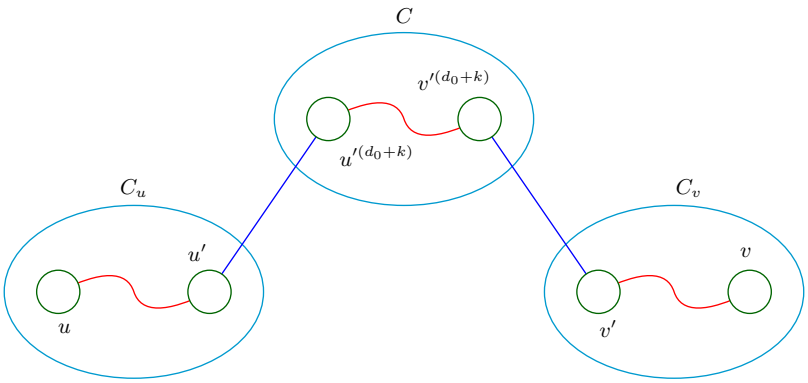

Figure 5: RDN_modified_routing

Lemma 4. *Algorithm 2 is correct for the sets V and W defined in Algorithm 4. The time complexity of Algorithm 2 is $O((D_0/\lg n_0)\lg N)$. The length of the path connects u to a node in V is at most $D_{k-1} = (D_0/2+1) * (\lg N + 1)/(\lg n_0 + 1) - 2$.*

Algorithm 2: RDN_one_path($RDN^k(B), u, V, W$)
Input: Node u and two sets of nodes, V and W, in $RDN^k(B)$ such that $C_u \cap V = \{v_1, \ldots, v_r\}$ and $r > 1$.
Output: Path $\{u \to v_a \in V\} \subset C_s$ and $r-1$ disjoint paths $v_i \to v'_i \notin C_{u_i}, 1 \le i \ne a \le r$, of length at most 2 such that $C_{v'_i} \cap W = \emptyset$ for all $i, 1 \le i \ne a \le r$, and $C_{v'_i} \ne C_{v'_j}$ if $i \ne j$
begin
  if $u \in V$ or $\exists i$ such that $u^{(i)} \in V$
  then $v_a = u$ or $u^{(i)}$
  else pick up v_a randomly;
    RDN_routing(C_u, u, v_a);
  endif
  find $r-1$ disjoint paths of length at most 2, $v_i \to v'_i, 1 \le i \ne a \le r$
  such that $C_{v'_i} \cap W = \emptyset$ for all $i, 1 \le i \ne a \le r$, and $C_{v'_i} \ne C_{v'_j}$ if $i \ne j$;
  if $\exists j$ such that $\{v_j \to v'_j\} \cap \{u \to v_a\} \ne \emptyset$
  then find v_b such that $\{v_b \to v'_b\} \cap \{u \to v_a\}$ closest to u;
    remove path $v_b \to v'_b$;
    replace path $u \to v_a$ by path $u \to v_b$;
    find a path of length at most 2, $v_a \to v'_a$ such that
      $C_{v'_a} \cap W = \emptyset$ and $C_{v'_a} \ne C_{v'_j}$ for all $j \ne a$;
  endif
end

Algorithm 3: RDN_modified_routing($RDN^k(B), u, v, C$)
Input: a cluster C and two nodes u and v in $RDN^k(B)$ such that $C_u \ne C_v$ and $type(C_u) = type(C_v) \ne type(C)$
Output: a path $u \to v$ that passes through C
begin
  RDN_routing(C_u, u, u'), where $(u')^{(d_0+k)} \in C$;
  RDN_routing(C_v, v, v'), where $(u')^{(d_0+k)} \in C$;
  RDN_routing($C, (u')^{(d_0+k)}, (v')^{(d_0+k)}$);
  return path $u \to v = u \to u' \to (u')^{(d_0+k)} \to (v')^{(d_0+k)} \to v' \to v$;
end

Proof. In Algorithm 4, $V = W = T$, initially, and if a node $v \in V$ is routed to $v' \in C_{v'} \ne C_v$ with $type(C_v \ne type(C_{v'})$ by a path of length at most 2 then $V = V \cup \{v'\} \setminus \{v\}$ and $W = W \cup \{v'\}$. Since the above routing and updating occurs at most once for every node in W, the number of nodes in W that reside in the clusters of a single type will never exceed $d_0 + k$. From Lemma 3, we know that, for each $v_i, 1 \le i \le r$, there are $d_0 + k$ disjoint paths of length at most 2 that route v_i to nodes in other distinct clusters. Therefore, there must exist a path $v_i \to v'_i$ such that $\{v_i \to v'_i\} \cap \{v_j \to v'_j, j \ne i\} = \emptyset$ for $j, 1 \le j \ne i \le r$ and $C_{v'} \cap W = \emptyset$. Since check the qualification of a path of length at most 2 takes only $O(1)$ time if we book-keep a boolean variable for each cluster C indicating whether $C \cap W = \emptyset$ or not. Therefore, the time complexity of Algorithm 2 is dominated by the time to find a single path inside the cluster which is $O(D_0 * 2^k) = O((D_0/\lg n_0)\lg N)$. The path $u \to V_a$ is of length at most D_{k-1}. □

Lemma 5. *Algorithm 3 is correct. The time complexity of Algorithm 3 is* $O((D_0/\lg n_0)\lg N)$. *The length of the paths is at most* $3(D_0/2+1)(\lg N+1)/(\lg n_0+1)-4$.

Proof. It is easy to see that Algorithm 3 finds a path from u to v that passes through clusters C_u, C, and C_v only. The time complexity of the algorithm is $O(3(2^{k-1} \times D_0 + 2^k) + 2) = O(2^k D_0) = O((D_0)/\lg n_0)\lg N$, and the length of the path is at most $3D_{k-1} + 2 = 3(2^{k-1} \times D_0 + 2^k - 2) + 2 = 3(2^{k-1} \times D_0 + 2^k) - 4 = 3(D_0/2+1)(\lg N+1)/(\lg n_0+1) - 4$. □

Now, we are ready to describe the proposed algorithm. The algorithm is divided into four stages to handle the routing in different situations. At the first stage, we handle the connection of those nodes in $T \cap C_s$ recursively if $0 < |T \cap C_s| < d_0 + k$. In case that $|T \cap C_s| = d_0 + k$, we pick up any $d_0 + k - 1$ nodes in T. Let the leftover node be t_a. We call the algorithm recursively on C_s to connect s with the $d_0 + k - 1$ nodes in T by disjoint paths in C_s. If any of the $d_0 + k - 1$ paths, say $s \rightarrow t_b$ meets t_a, we replace path $s \rightarrow t_b$ by path $s \rightarrow t_a$ and let the leftover node be t_b. Then, the leftover node of T can be connected to s via a path that is completely outside the cluster C_s except node s and the leftover node of T.

After the recursive call to connect the nodes in $T \cap C_s$, we route the node(s) in $N(s) \cap C_s$ that is (are) not used yet out of C_s by cross-edge(s). Let these nodes $s^{(i,d_0+k)}, i \neq d_0 + k$, and node $s^{(d_0+k)}$ be $U = \{u_1, \ldots, u_q\}$, $V = T \setminus C_s = \{v_1, \ldots, v_q\}$, and $W = T$. In Stages 2 - 4, the nodes in U and nodes in V will be connected via distinct clusters. The set W is used while selecting the destination cluster for the path $v_i \rightarrow v'_i$ of length at most 2. That is, a destination cluster for v'_i should not contain any node in W.

At the second stage, we handle only the clusters $C_{u_i}, 1 \leq i \leq q$, with $C_{u_i} \cap T \neq \emptyset$ and the clusters C with $type(C) \neq type(C_s)$ and $|C \cap V| > 1$. We call N2S_one_path to connect u_i to a node v_a in $V \cap C_{u_i}$ and route nodes $v_j, j \neq a$, to v'_j via disjoint paths of length at most 2 such that $C_{v'_j}$ are all distinct and $C_{v'_j} \cap V = \emptyset$. For the clusters C with $type(C) \neq type(C_s)$ and $|C \cap V| > 1$, we spread all nodes in $C \cap V$ out of C via disjoint paths of length 2 similarly. Then, we update U and V by removing the pair nodes in C_{u_i} that are connected and those nodes v_j in V that are routed to $v'_j \notin C_{v_j}$ from U and V, and adding v'_j to V and W.

At the third stage, we handle those clusters C with $type(C) = type(C_s)$ and $C \cap V \neq \emptyset$. For each of these clusters, we route a node $u \in U$ to a node $w \in C$, where path $u \rightarrow w^{(d_0+k)} \subset C_u$. Notice that $C_u \cap V = \emptyset$ after the 2nd stage, the path $u \rightarrow w^{(d_0+k)}$ can be generated by RDN_routing in C_u. Then, we call RDN_routing (if $|C \cap V| = 1$) or RDN_N2S_one_path (if $|C \cap V| > 1$) to connect w to a node $v_a \in V \cap C$ and route all other nodes $v_j, j \neq a$ in $V \cap C$ to node v'_j in distinct clusters that does not contain any node in V by disjoint paths of length at most 2. Then, we update U and V accordingly as in Stage 2 after the routing.

Finally, at the fourth stage, we handle clusters C with $type(C) \neq type(C_s)$ that contain nodes in $U \cup V$ again. Notice that, after Stage 3, C_v will be distinct for every $v \in V$, and $C_v \cap U = \emptyset$ or $|C_v \cap U| = 1$. The case $|C_v \cap U| = 1$ is trivial (just connect them in C_v by RDN_routing). If $C_v \cap U = \emptyset$, we pick up any u with $C_u \cap V = \emptyset$ and find an intermediate cluster C with $type(C) = type(C_s)$ and that does not contain s or any node in T or any v'_j generated in Stage 2. Then, we call MRDN_modified_routing to connect u to v via cluster C.

The proposed algorithm is formally presented as Algorithm 4. The correctness of Algorithm 4 and the upper bound of the length of disjoint paths are given in the main theorem, Theorem 1, of the paper.

Theorem 1. *Assume that d_0 and D_0 are the node-degree and the diameter of the base network B, respectively. Assume that d_0 disjoint paths exist in B between a node and a set of d_0 nodes in B. Let s be a node and T a set of $d_0 + k$ nodes in $RDN^k(B), k > 0$. Then $d_0 + k$ disjoint paths, each connecting s to a node in T, can be found in $O(((d_0 + k)D_0/\lg n_0) \lg N)$ time. The length of the paths is at most $3(D_0/2 + 1)(\lg N + 1)/(\lg n_0 + 1)$.*

Proof. In Stage 1, there are two cases: $r < d_0 + k$ and $r = d_0 + k$. Case 1 is trivial following the induction on k. In Case 2, $d_0 + k - 1$ disjoint paths can be generated in C_s. Let the unconnected node in T be t_a. Then, it is easy to see that $s^{(d_0+k)}$ and $t_a^{(d_0+k)}$ can be connected completely outside C_s. In Stage 2, there are two types of routings: one for cluster C_u with $C_u \cap V \neq \emptyset$ and the other for the cluster C with $type(C) \neq type(C_s)$ and $|C \cap V| > 1$. The argument of the correctness of the routing at Stage 2 is similar to that in the proof of Lemma 4. However, for the routing of second type, although there exists a path of length at most 2, $v_j \rightarrow v'_j$, for each $v_j \in V \cap C$ such that it is disjoint with other paths of length at most 2 and $C_{v'_j} \cap V = \emptyset$ due to the fact $|V \setminus \{v_j\}| < d_0 + k$, $v'_j \in C_s$ might occur. If it occurs then we have $C_s \cap T = \emptyset$. In such case, we can connect v'_j with s in C_s and then remove the neighbor of s in the path from U.

Algorithm 4: RDN_N2S_disjoint_paths($RDN^k(B), s, T$)
Input: Node s and a set T of $m \leq d_0 + k$ nodes in $RDN^k(B), k > 0$
Output: m disjoint paths connecting node s and $t_i \in T, 1 \leq i \leq m$
begin
$U \leftarrow N(s)$; $V = W \leftarrow T$;
if $|C_s \cap V| = r > 0$ /* Stage 1 */
then if $r < d_0 + k$
then if $k = 1$ **then** RDN_N2S_disjoint_paths(B, u, v)
else RDN_N2S_disjoint_paths($RDN^{k-1}(B), s, V \cap C_s$);
for each $i, 1 \leq i < d_0 + k$, **do**
if $s^{(i)}$ in path $s \rightarrow v_{j_i}$ for some $v_{j_i} \in C \cap V$
then $U \leftarrow U \setminus \{s^{(i)}\}$;
$V \leftarrow V \setminus \{v_{j_i}\}$;
else $U \leftarrow U \cup \{s^{(i,d_0+k)}\} \setminus \{s^{(i)}\}$;
else pick up any node, say $v_0 \in V$;
RDN_N2S_disjoint_paths($RDN^{k-1}(B), s, V \setminus \{v_0\}$);
if any of the $d_0 + k - 1$ paths, say $s \rightarrow v_j, j > 1$, meets v_0
then replace $s \rightarrow v_j$ by $s \rightarrow v_j$;
$V \leftarrow \{v_j^{(d_0+k)}\}$;
else $V \leftarrow \{v_0^{(d_0+k)}\}$;
$U \leftarrow \{s^{(d_0+k)}\}$;
for each $u \in U$ with $C_u \cap V \neq \emptyset$ **do** /* Stage 2 */
if $|C_u \cap V| = 1$ **then** RDN_routing($RDN^k(B), u, v$)
else RDN_N2S_one_path($RDN^k(B), u, V, W$);
$U \leftarrow U \setminus \{u\}$;
$V \leftarrow V \cup \{v'_i, 1 \leq i \neq a \leq r\} \setminus \{v_i, 1 \leq i \leq r\}$;
$W \leftarrow W \cup \{v'_i, 1 \leq i \neq a \leq r\}$;
for each C with $type(C) \neq type(C_s)$ and $|C \cap V| = m > 1$ **do** /* Let $C \cap V = \{v_1 \ldots, v_m\}$. */
find m disjoint paths of length at most 2, $v_i \rightarrow v'_i, 1 \leq i \leq m$,
such that $C_{v'_i} \cap (T \cup V) = \emptyset$ and $C_{v'_i} \neq C_{v'_j}$ if $i \neq j$;
$V \leftarrow V \cup \{v'_i, 1 \leq i \leq m\} \setminus \{v_i, 1 \leq i \leq m\}$;
$W \leftarrow W \cup \{v'_i, 1 \leq i \leq r\}$;
while $\exists$ cluster C with $type(C) = type(C_s)$ and $|C \cap V| > 0$ **do** /* Stage 3 */
find a $u \in U$;
route u to a $w \in C_u$ such that $w^{(d_0+k)} \in C$;
if $|C_u \cap V| = 1$ **then** RDN_routing($RDN^k(B), u, v$)
else RDN_N2S_one_path($RDN^k(B), u, V, W$);
$U \leftarrow U \setminus \{u\}$;
$V \leftarrow V \cup \{v'_i, 1 \leq i \neq a \leq r\} \setminus \{v_i, 1 \leq i \leq r\}$;
$W \leftarrow W \cup \{v'_i, 1 \leq i \neq a \leq r\}$;
for each node $u \in U$ **do** /* Stage 4 */ /* $type(C_v) = type(C_u)$ */
if $\{C_u \cap V\} = v$ **then** RDN_routing(C_u, u, v)
else pick up a $v \in V$;
find a cluster C s.t. $type(C) = type(C_s)$ and $C \cap W \neq \emptyset$;
RDN_modified_routing($RDN^k(B), u, v, C$);
$U \leftarrow U \setminus \{u\}$;
$V \leftarrow V \setminus \{v\}$;
end

In Stage 3, we connect $u \in U$ with a $v \in V$, where $|C_v \cap V| > 0$ and $type(C_v) = type(C_s)$, one u for each cluster C_v with $C_v \cap V \neq \emptyset$. Since $|V| \leq d_0 + k$ in any case, the argument for the correctness of this stage is again similar to that in the proof of Lemma 4. However, when call RDN_one_path for

C_v, the node v'_j that is generated by distributing v_j via a path of length at most 2 might be in C_u for a unconnected $u \in U$. If this occurs, we simply connect v'_j with u in C_u. Otherwise, connection of v'_j with an unconnected u will be handled by calling RDN_modified_routing (see Stage 4).

In Stage 4, it is easy to see from the way we updated V in Stages 1 - 3 that, for every $v \in V$, we have $V \cap C_v = \{v\}$ and $type(C_v) \neq type(C_s)$. If $C_v \cap U = \{u\}$, it is trivial to connect them inside C_v. Otherwise, we connect v with a node u pairwisely via a cluster C with $type(C) = type(C_s)$, $C_v \neq C_s$, and that is not used for the connections in Stage 3. Let the number of the node pairs to be connected be $r > 0$. We show that there are at least r clusters satisfying the conditions. Since the number of clusters C' with $type(C') \neq type(C_s)$ is at least $d_0 + k + r$ since all C_v are distinct from all C_u. That is, the number of clusters of type C_s is also at least $d_0 + k + r$. The number of the clusters used in Stage 3 is at most $d_0 + k - r$. Therefore, we have at least $(d_0 + k + r) - (d_0 + k - r) - 1 = 2r - 1 \geq r$ clusters of the same type of C_s and excluding C_s that can be used as intermediate clusters in Stage 4. Therefore, every pair of u and v that are in the distinct clusters can be connected by calling RDN_modified_routing.

It is clear from the algorithm that the time complexity of the algorithm is dominated by routing in stages 2 - 4. From Lemmas 3 - 5, the time complexities of Algorithms 1 - 3 are all $O(2^k D_0)$. These algorithms are called at most $m \leq d_0 + k$ times in Algorithm 4. Since the operations in Algorithm 4 for distributing nodes in $N(s)$ and T from a cluster to another cluster take $O(d_0 + k)$ time, the time complexity of Algorithm 4 is $O((d_0 + k)2^k D_0) = O(((d_0 + k)D_0 / \lg n_0) \lg N)$. The longest path among the $m \leq d_0 + k$ disjoint paths is the one in Stage 4, which is of length at most $3D_{k-1} + 6 = 3(2^{k-1}D_0 + 2^k - 2) + 6 = 3(D_0/2 + 1)2^k) = 3(D_0/2 + 1)(\lg N + 1)/(\lg n_0 + 1)$. □

We give an example of node-to-set disjoint-paths routing in $RDN^2(B)$ that follows the proposed algorithm step-by-step.

Example 1: In $RDN^2(B)$, where B is a ring with 3 nodes, let $s = (0, (0, 0, 0), (0, 0, 0))$, and $t_1 = (0, (1, 0, 0), (1, 1, 0))$; $t_2 = (0, (1, 0, 0), (0, 1, 1))$;
$t_3 = (1, (1, 1, 1), (1, 0, 0))$; and $t_4 = (1, (1, 1, 1), (0, 2, 2))$.

For simplicity, we do not include the updating of set W. Basically, any nodes generated from a distributing path of length at most 2 from a node in T should be added to W. The clusters containing any node in W cannot be selected as destination clusters while distributing the nodes in a cluster that contains more than one node in T.

• Stage 1: Since $C_s \cap V = \emptyset$, $s^{(1)}$, $s^{(2)}$, and $s^{(3)}$ are routed out of C_s via cross-edges of level 2.
The nodes in U are updated and $U = \{(1, (0, 0, 1), (0, 0, 0)), (1, (0, 0, 2), (0, 0, 0)),$
$(1, (1, 0, 0), (0, 0, 0)), (1, (0, 0, 0), (0, 0, 0))\}$ (see Figure 6).

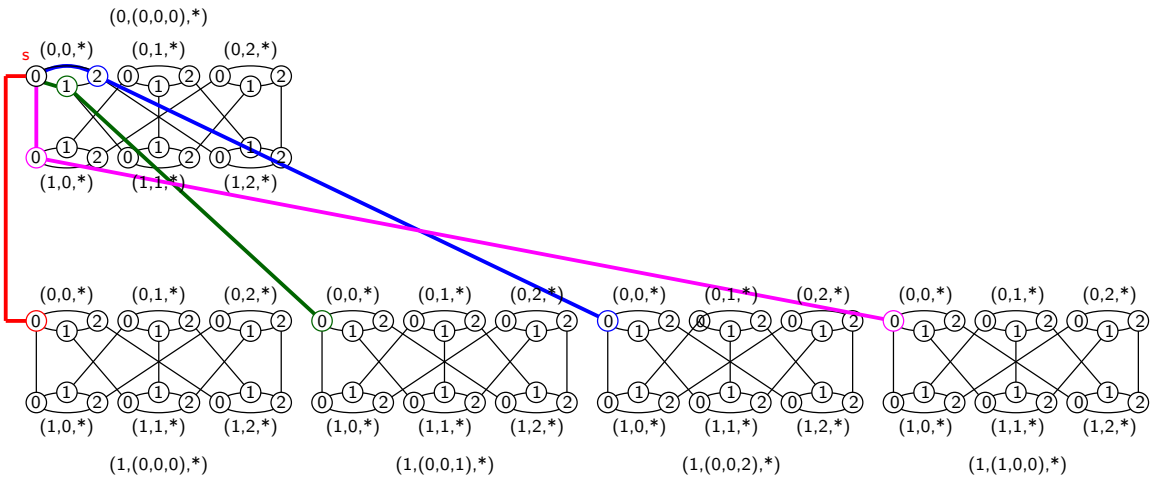

Figure 6: Routing at stage 1

• Stage 2: Since t_3 and t_4 are in the same cluster of type 1, we route them out of the cluster by disjoint paths of length at most 2.

The nodes in V are updated and

$V = \{t_1, t_2, (0,(1,0,1),(1,1,1)), (0,(0,2,2),(1,1,1))\}$.

Notice that since $t_3^{(4)} \in C_{t_1}$, we need a path of length 2 to route t_3 out of the cluster.

• Stage 3: Since there are 3 clusters of type 0 that contain nodes in V, we pick up any 3 nodes in U and route each of them to one of the 3 clusters. Then, we call RDN_routing or RDN_N2S_one_path for each of the 3 clusters. After Stage 3, three disjoint paths are generated and t_2 is routed to $(1,(0,1,1),(1,0,0))$ via a cross-edge of level 2 (see Figure 7).

Path #1: $(1,(0,0,1),(0,0,0)) \Rightarrow (1,(0,0,1),(1,0,1)) \rightarrow (0,(1,0,1),(0,0,1)) \Rightarrow (0,(1,0,1),(1,1,1))$;

Path #2: $(1,(0,0,2),(0,0,0)) \Rightarrow (1,(0,0,2),(0,2,2)) \rightarrow (0,(0,2,2),(0,0,2)) \Rightarrow (0,(0,2,2),(1,1,1))$; and

Path #3: $(1,(1,0,0),(0,0,0)) \Rightarrow (1,(1,0,0),(1,0,0)) \rightarrow (0,(1,0,0),(1,0,0)) \Rightarrow (0,(1,0,0),(1,1,0))$.

Finally, we update U, V. Both sets contain only single nodes. $U = \{(1,(0,0,0),(0,0,0)) = s^{(4)}\}$ and $V = \{(1,(0,1,1),(1,0,0))\}$.

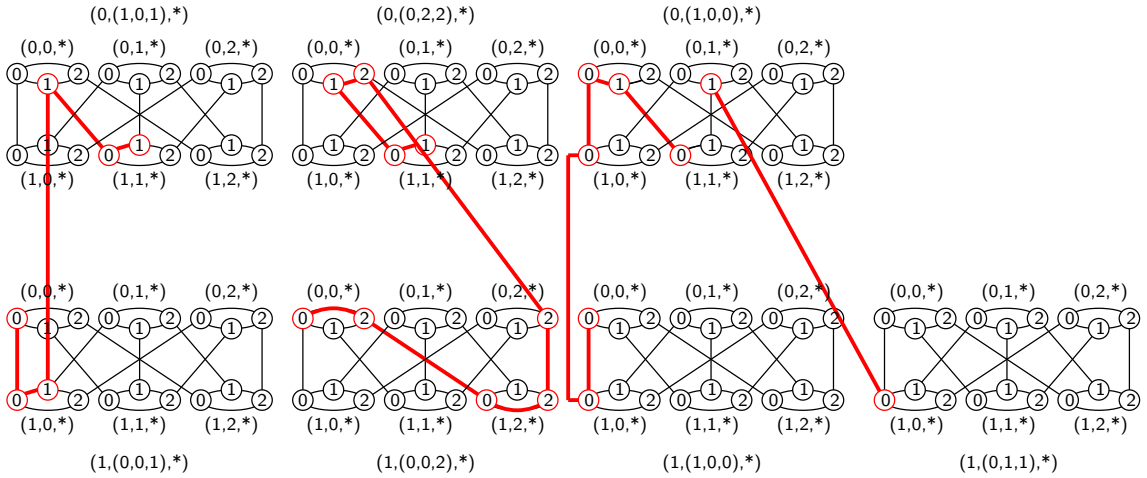

Figure 7: Routing at stage 3

• Stage 4: We pick up the cluster C of type 0 whose cluster ID = (0,0,1) because $C \cap W = \emptyset$. We call RDN_Modified_Routing to connect the last pair of nodes in U and V via cluster C (see Figure 8).

Path #4: $(1,(0,0,0),(0,0,0)) \Rightarrow (1,(0,0,0),(0,0,1)) \rightarrow (0,(0,0,1),(0,0,0)) \Rightarrow (0,(0,0,1),(0,1,1)) \rightarrow (1,(0,1,1),(0,0,1)) \Rightarrow (1,(0,1,1),(1,0,0))$.

4 Concluding Remarks

Recursive dual-net is a potential interconnection network for supercomputers of next generations because of its low node degree and short diameter for the extremely large parallel computer systems. Its symmetric and recursive structure, and simple routing algorithms are also attractive. In this paper, we proposed an efficient algorithm for node-to-set disjoint-paths routing on recursive dual-net. There are many other interesting and fundamental communication and computational problems on recursive dual-net that are worth further research. For example, in parallel programming, collective communication is generally implemented in a way that routing messages from a set of nodes to another set of nodes so they don't interfere with each other. Therefore, providing an efficient algorithm for set-to-set disjoint-paths routing is an important and interesting future research work.

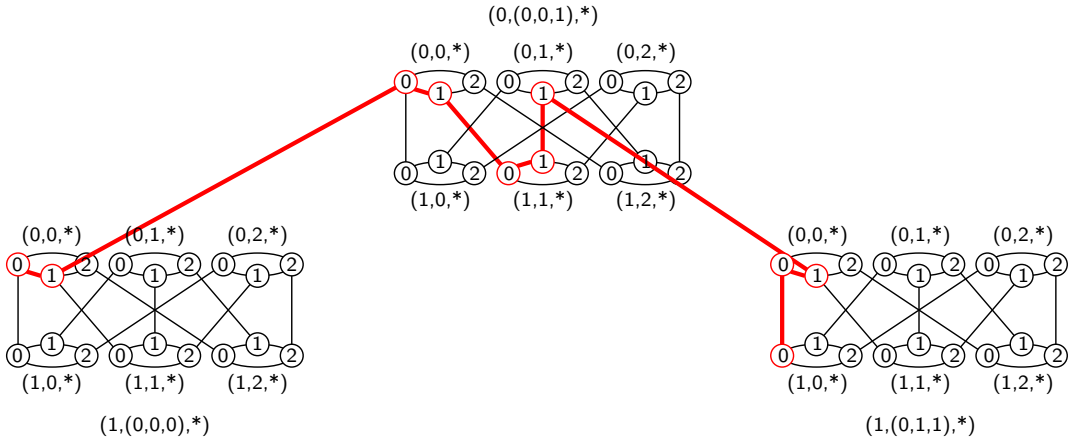

Figure 8: Routing at stage 4

References

[1] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue gene/l torus interconnection network. *IBM Journal of Research and Development, http://www.research.ibm.com/journal/rd/492/tocpdf.html*, 49(2/3):265–276, 2005.

[2] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, Apr. 1989.

[3] P. Beckman. Looking toward exascale computing. In *Proceedings of the Proceedings of the 2008 International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dec. 2008. keynote speaker.

[4] A. Bossard, K. Kaneko, and S. Peng. Node-to-set disjoint-paths routing in metacube. In *Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 8–11, Hiroshima, Japan, Dec. 2009. IEEE Computer Society Press.

[5] P. F. Corbett. Rotator graphs: An efficient topology for point-to-point multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):622–626, May 1992.

[6] K. Day and A. Tripathi. A comparative study of topological properties of hypercubes and star graphs. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):31–38, Jan. 1994.

[7] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, 1997.

[8] Q.-P. Gu and S. Peng. Set-to-set fault tolerant routing in star graphs. *IEICE Trans. on Information and Systems*, E79-D(4):282–289, 1996.

[9] Q.-P. Gu and S. Peng. Node-to-set disjoint paths problem in star graphs. *Information Processing Letters*, 62(4):201–207, May 1997.

[10] Q.-P. Gu and S. Peng. Node-to-set and set-to-set cluster fault tolerant routing in hypercubes. *Parallel Computing*, 24(9):1245–1261, Aug. 1998.

[11] Z. Jovanovic and J. V. Misic. Fault tolerance of the star graph interconnection network. *Information Processing Letters*, 49(3):145–150, Feb. 1994.

[12] K. Kaneko and Y. Suzuki. Node-to-set disjoint paths problem in pancake graphs. *IEICE Transactions on Information and Systems*, E86-D(9):1628–1633, Sep. 2003.

[13] S. Latifi. On the fault-diameter of the star graph. *Information Processing Letters*, 46(3):143–150, Jun. 1993.

[14] Y. Li and S. Peng. Dual-cubes: a new interconnection network for high-performance computer clusters. In *Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture*, pages 51–57, ChiaYi, Taiwan, December 2000.

[15] Y. Li, S. Peng, and W. Chu. Metacube - a new interconnection network for large scale parallel systems. *Australian Computer Science Communications*, 24(3):29–36, Jan. 2002.

[16] Y. Li, S. Peng, and W. Chu. Disjoint-paths and fault-tolerant routing on recursive dual-net. In *Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 48–56, Hiroshima, Japan, Dec. 2009. IEEE Computer Society Press.

[17] Y. Li, S. Peng, and W. Chu. Recursive dual-net: A new versatile network for supercomputers of the next generation. *Journal of Chinese Institute of Engineer*, 32:931–938, Nov. 2009.

[18] Y. Suzuki and K. Kaneko. An algorithm for node-disjoint paths in pancake graphs. *IEICE Transactions on Information and Systems*, E86-D(3):610–615, Mar. 2003.