International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 4, Number 2, pages 260–278, July 2014

Disjoint-Path Routing on Hierarchical Dual-Nets

Jun Arai
Graduate School of CIS
Hosei University
Tokyo 184-8584 Japan

Yamin Li
Faculty of Computer and Information Sciences
Hosei University
Tokyo 184-8584 Japan

Received: February 14, 2014
Revised: May 3, 2014
Accepted: May 30, 2014
Communicated by Susumu Matsumae

Abstract

The hierarchical dual-net (HDN) was introduced as a topology of interconnection networks for extremely large parallel computers. The HDN is constructed based on a symmetric product graph (base network). A k-level hierarchical dual-net, HDN(B, k, S), contains $(2N_0)^{2^k}/(2 \times \prod_{i=1}^{k} s_i)$ nodes, where $S = \{G'_1, G'_2, \ldots, G'_k\}$, G'_i is a super-node and $s_i = |G'_i|$ is the number of nodes in the super-node at the level i for $1 \leq i \leq k$, and N_0 is the number of nodes in the base network B. The node degree of HDN(B, k, S) is $d_0 + k$, where d_0 is the node degree of the base network. The HDN is node and edge symmetric and can contain huge number of nodes with small node-degree and short diameter. Disjoint-path routing is a fundamental and critical issue for the performance of an interconnection network. In this paper, we propose an efficient algorithm for finding disjoint-paths on an HDN and give the performance simulation results.

Keywords: Interconnection network, routing algorithm, disjoint paths

1 Introduction

Recently, because of the advances in computer and networking technologies, supercomputers containing hundreds of thousands of nodes have been built [9]. It was predicted that the parallel systems of the next decade will contain 10 to 100 millions of nodes. The interconnection network plays an important role for achieving high-performance in such ultra-scale parallel systems. The performance of an ultra-scale parallel computers depends largely on the time complexities of communication schemes, and in turn depends on the diameter of the network.

An interconnection network consists of switches with multiple communication ports and cables connecting ports by following certain topologies. For an ultra-scale parallel computer, the traditional interconnection networks may no longer satisfy the requirements for the high-performance computations or efficient communications. For such an ultra-scale parallel computer, the node degree and the diameter will be the critical measures for the effectiveness of the interconnection networks. The node degree is limited by the hardware technologies and the diameter affects all kinds of communication schemes directly. The number of communication ports (node degree) in the network-on-chip (NoC) is typically 4 to 8 in current implementations. The off-chip interconnect switches can have tens of ports, but the cost becomes expensive as the number of ports increases. Other important measures for the effectiveness of the interconnection networks include symmetricity, scalability, and efficient routing algorithms.

The following two categories of interconnection networks have attracted a great research attention and been used in many supercomputers' implementations. One is the hypercube-like family that has the advantage of short diameters for high-performance computing and efficient communications [8]. The other is the 2D/3D mesh or torus family that has the advantage of small and fixed node degrees and easy implementations [1]. Traditionally, most supercomputers including those built by CRAY, IBM, SGI, and Intel use 3D tori or hypercubes.

However, the node degree of the hypercube increases logarithmically as the number of nodes in the systems increases; the diameter of the 2D/3D torus becomes large in an ultra-scale parallel system. To solve these problems, the hierarchical (cluster-based) architectures are proposed in literature [2, 4, 7]. The supercomputers built by IBM recently, Roadrunner, adopt a new approach for the interconnection network [3]. It is a cluster-based architecture: the connection among clusters is fully connected, and the fat-tree is used for the connection inside a cluster.

In this paper, we first present a flexible interconnection network, called *Hierarchical Dual-Net* (HDN) [6]. The HDN is symmetric and can connect a large number of nodes with a small node degree, meanwhile keeping the diameter short. The HDN was motivated by recursive dual-net (RDN) [5]. The RDN has merits of low node degree and short diameter. The problem of the RDN is that it grows too fast in size, and there is no mechanism to control the rate of its growth. Different from the RDN, the scale of the HDN can be controlled by setting a set of suitable parameters while generating an expanded network through dual-construction. The HDN also adapts to the cluster-based architecture. Compared to the Roadrunner, the HDN is symmetric, uses small number of links, and meanwhile keeps the diameter short. The HDN structure is also better than other popular existing networks such as hypercube and 2D/3D torus with respect to the degree and diameter. We investigate the topological properties of the HDN and show some examples of HDNs with simple base networks of small size. Then we compare them to other networks such as three-dimensional torus used in IBM Blue Gene/L [1], and hypercube [8].

The main contribution of this paper is the disjoint-path routing algorithm on hierarchical dual-net. Let d_0 be the node-degree of the symmetric base-network B. Given two nodes s and t in a hierarchical dual-net HDN(B, k, S) with a base network B such that, for any two nodes in B, there are d_0 disjoint-paths connecting them in $O(d_0^2)$ time, we propose an $O((d_0 + k)2^k)$ time algorithm for finding $d_0 + k$ disjoint-paths connecting s and t.

The rest of this paper is organized as follows. Section II introduces the hierarchical dual-net in detail. Section III describes the disjoint-path routing algorithms on a hierarchical dual-net. Section IV gives the performance evaluation results. Section V concludes the paper.

2 The Hierarchical Dual-Net

We begin with a brief introduction to the recursive dual-net (RDN) [6], the details of the RDN descriptions can be found in [6]. An RDN is constructed recursively by a dual-construction. The dual-construction is a way to expand a given symmetric graph G of size n to a new symmetric graph G^* of size $2n^2$. It generates $2n$ copies of G as subgraphs (denoted as clusters) of G^*. Half of them, n clusters, are of class 0 and the others are of class 1.

If G is symmetric then the expanded graph G^* is unique and symmetric. Therefore, the dual-construction can be applied recursively from a symmetric network (the base network). $RDN(m, k)$ denotes an RDN generated from a base network of size m by applying dual-construction k times.

The problem about an RDN is that its growth rate is super-exponential ($(2m)^{2^k}/2$). There is very little space for selection of the size of an RDN. For example, let the base network be a 3-cube, then the sizes of $RDN(8,k)$ will be 2^7, 2^{15}, and 2^{31} for $k = 1, 2$, and 3, respectively. In HDN, we provide a mechanism to control the growth rate through its expansion from a base network. This new interconnection network has a very flexible way for adjusting its size.

The *hierarchical dual-net*, HDN(B,k,S), contains three sets of parameters: B is a *symmetric product graph*, we call it *base network*; k is an integer that indicates the *level* of the HDN (the number of *dual-constructions* applied); and $S = \{G'_1, G'_2, \ldots, G'_k\}$, where G'_i is a *sub-graph* of HDN(B,k,S) and $s_i = |G'_i|$ is the number of nodes in a *super-node* at the level i for $1 \le i \le k$. All these terminologies will be defined in the following paragraphs.

Given r graphs $G_i = (V_i, E_i)$, $1 \le i \le r$, their product graph $G = G_1 \times G_2 \times \ldots \times G_r$ is defined as the graph $G = (V,E)$, where $V = \{(v_{j1}, \ldots, v_{ji}, \ldots, v_{jr}) | v_{ji} \in V_i,\ 1 \le i \le r\}$ and $E = \{[(v_{j1}, \ldots, v_{ji}, \ldots, v_{jr}), (v_{k1}, \ldots, v_{ki}, \ldots, v_{kr})] | v_{ji} \ne v_{ki},\ (v_{ji}, v_{ki}) \in E_i$, and $v_{jl} = v_{kl}$ for $l \ne i, 1 \le i \le r\}$. Given a product graph $G = G_1 \times G_2 \times \ldots \times G_r$, we define a *quotient graph* Q as $Q = G/G'$ where G' is a sub-product graph of G such that $G = G' \times Q$. A node in a product graph $G = G_1 \times \ldots \times G_i \times \ldots \times G_r$ can be represented by $(a_1, \ldots, a_i, \ldots, a_r)$ with $0 \le a_i \le |G_i| - 1$. We define a sub-graph G' as $G' = G''_1 \times \ldots \times G''_j \times \ldots \times G''_q$ with $G''_j = G_i$ for $1 \le j \le q \le r$ and $1 \le i \le r$, $G''_j \ne G''_k$ if $j \ne k$ for $1 \le j, k \le q$. Then a node in the sub-graph G' can be represented by $(b_1, \ldots, b_i, \ldots, b_q)$ with $0 \le b_i \le |G''_i| - 1$. We can consider a quotient graph Q as a reduced graph of G with G' being mapped into a single node (a super-node).

A graph G is symmetric (node-symmetric) if all its nodes looks alike. A product graph is symmetric if all its component graphs are symmetric. We use the symmetric product graph as the base network for generating a hierarchical dual-net through dual-constructions. We denote the base network as $B = B_1 \times B_2 \times \ldots \times B_r$ where all the B_i, $1 \le i \le r$, are symmetric. We define a super-node of B, denoted as SN as a sub-product graph of B. That is, $SN = B_{i_1} \times B_{i_2} \times \ldots \times B_{i_q}$, where $i_j, 1 \le j \le q$, are distinct and $q \le r$. Let $|B_i| = b_i$ be the number of nodes in B_i for $1 \le i \le r$. The HDN$(B,0,S) = B$ is the base network. For $i > 0$, the HDN(B,i,S) is generated from HDN$(B,i-1,S)$ by a construction to be explained below. Note that $S = \{G'_1, G'_2, \ldots, G'_k\}$, where G'_i is a sub-graph of HDN$(B,k-1,S)$ and $s_i = |G'_i|$ is the number of nodes in a super-node at the level i for $1 \le i \le k$. First, we define a super-node of level i, denoted as SN^i, to be a sub-product graph G'_i of size s_i in B. Then, we define graph Q^i as the quotient graph HDN$(B,i-1,S)/SN^i$. Suppose that there are N_{i-1} nodes in the HDN$(B,i-1,S)$, then the number of nodes n_i in Q^i is N_{i-1}/s_i. The s_i can be 1 or $\prod_{j=1}^{q} |B_{i_j}|$, where $1 \le i_j \le r$ and $q \le r$. That is, s_i can be a product of any number of integers in $\{b_1, b_2 \ldots, b_r\}$. For example, if $r = 3$, $b_1 = 2$, $b_2 = 3$, and $b_3 = 5$, the possible s_i can be 1, 2, 3, 5, 2×3, 2×5, 3×5, or $2 \times 3 \times 5$.

The construction of HDN(B,i,S), $1 \le i \le k$, can be defined by a two-step process: First, we perform a dual-construction on the quotient graph $Q^{i-1} =$ HDN$(B,i-1,S)/SN^i$ (HDN$(B,0,S)$= B). Let the graph generated by the dual-construction be Q^i, and the subgraph of two nodes that is connected by a cross-edge of level i be K_2. Second, to get the HDN(B,i,S), we replace every K_2 in Q^i by a product graph $K_2 \times SN$. We call HDN$(B,i-1,S)$ *cluster* of HDN(B,i,S).

Referring to Figure 1, an HDN(B,i,S) consists of $2n_i$ clusters which are divided into two *classes*: class 0 and class 1 with each class containing n_i clusters. That is, the number of clusters in each class is equal to the number of super-nodes in a cluster. At level i, each super-node in a cluster has s_i new links to a super-node in a distinct cluster of the other class. Because there are s_i nodes in a super-node, one node contributes a new link. The dual-construction of an RDN is a special case of the construction of an HDN with $s_i = 1$ for $1 \le i \le k$.

The indexes of the nodes in HDN(B,k,S) can be defined as follows. Let SN^k_{id} be a *super-node_id* in a cluster of HDN(B,k,S) and N^k_{id} be a *node_id* in a super-node, then a node in the HDN(B,k,S) can be represented by $(C^k, U^k_{id}, SN^k_{id}, N^k_{id})$ where C^k is the *class_id* (0 or 1) and U^k_{id} is the *cluster_id*. A cross-edge at level k connects node $(C^k, U^k_{id}, SN^k_{id}, N^k_{id})$ and node $(\overline{C^k}, SN^k_{id}, U^k_{id}, N^k_{id})$, reverting C^k and exchanging U^k_{id} and SN^k_{id}.

Two HDN examples are shown in Figures 2 and 3, where the base network is a 2-cube. Figure 2 shows an HDN$(B,1,S)$ with $s_1 = 2$. There are 2 super-nodes (SN 0 and SN 1) in a cluster and each

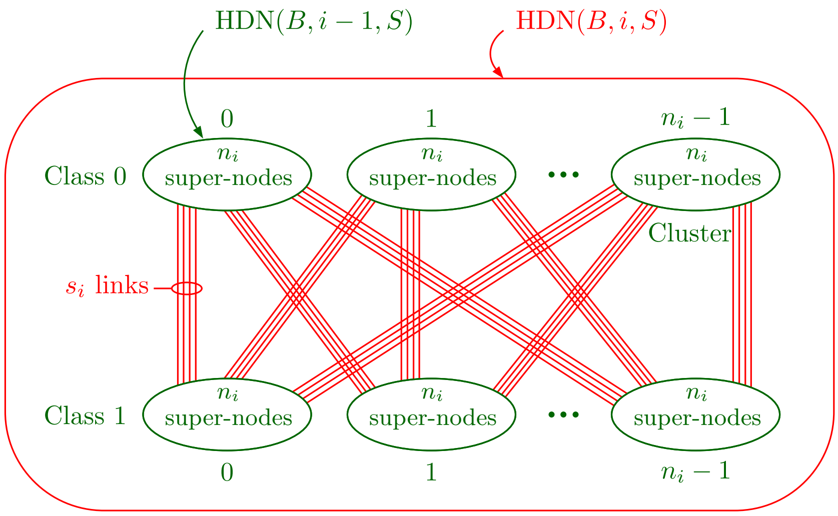


Figure 1: Build an HDN(B, i, S) from HDN$(B, i-1, S)$ [6]

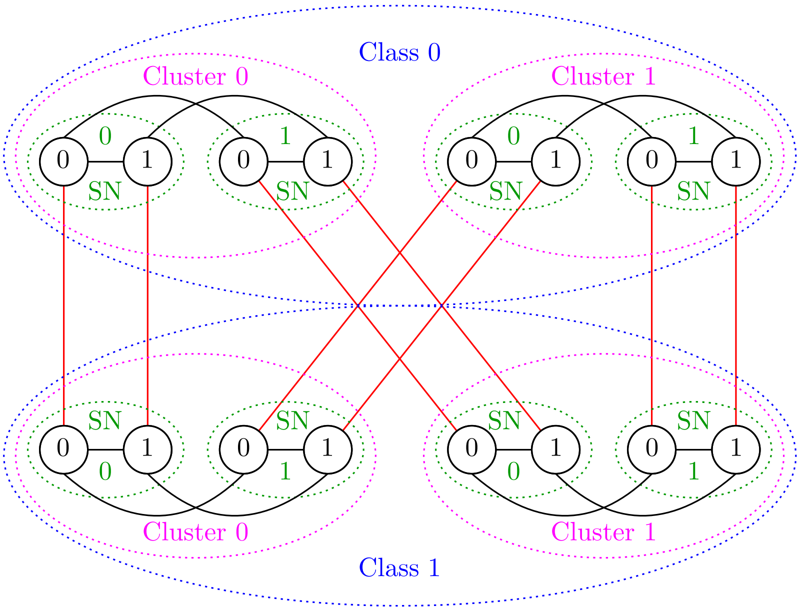


Figure 2: An HDN$(B, 1, S)$ with $s_1 = 2$ [6]

contains 2 nodes: node 0 and node 1. Each class has 2 clusters (the number of clusters in a class is equal to the number of super-nodes in a cluster). Figure 3 shows an HDN$(B, 2, S)$ with $s_2 = 4$, also based on HDN$(B, 1, S)$. More details of HDN, such as the number of nodes and topological properties, are presented in [6]. The following theorem summarizes some properties of the HDN.

Theorem 1 *Assume that the base network B is a symmetric, product graph and $SN^i, 1 \leq i \leq k$, are sub-product graphs of B with $|SN^i| = s_i$. Let the number of nodes, the node-degree, and the diameter of B be N_0, d_0, and D_0, respectively. Let the diameters of $SN^i, 1 \leq i \leq k$, be $D(SN^i)$. Let $S = \{G'_1, G'_2, \ldots, G'_k\}$, where G'_i is a sub-graph of HDN$(B, k-1, S)$ and $s_i = |G'_i|$ is the number of nodes in a super-node at the level i for $1 \leq i \leq k$. Then, the number of nodes of HDN(B, k, S) is $(2N_0)^{2^k}/(2\prod_{i=1}^{k} s_i)$, the node-degree is $d_0 + k$, and the diameter is $D_k = 2^k D(B) - \sum_{j=0}^{k-1} 2^j D(SN^{k-j}) + 2^{k+1} - 2$, where N is the number of nodes in HDN(B, k, S).*

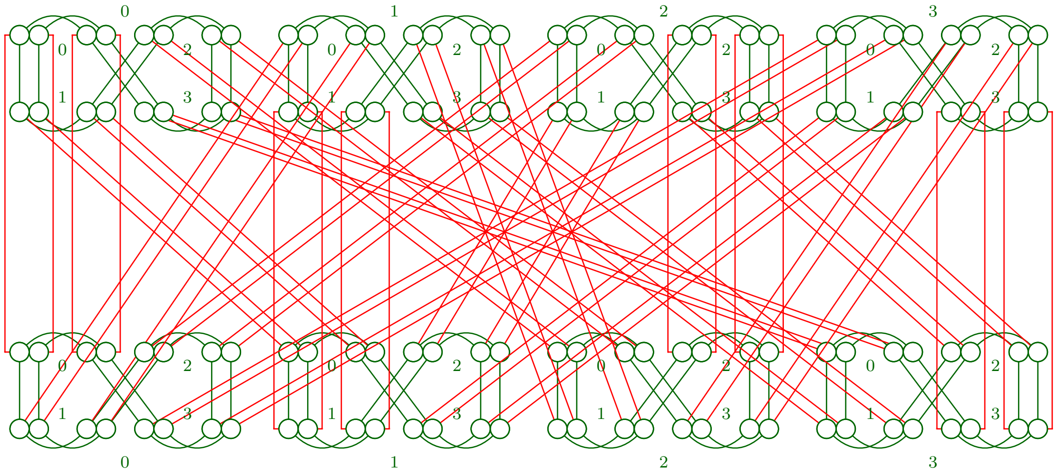


Figure 3: An HDN$(B, 2, S)$ with $s_1 = 2$ and $s_2 = 4$ [6]

3 Disjoint-path Routing on HDN

The problem of finding a path from a source s to destination t and forwarding a message along the path is known as the routing problem. Finding multiple, disjoint-paths for routing from s to t is called disjoint-path routing. The solutions for these routing problems are fundamental and critical for the performance of an interconnection network.

In this section, we introduce an efficient routing algorithm and propose a disjoint-path routing algorithm that finds d disjoint-paths on a hierarchical dual-net with the node degree of d.

3.1 Routing on HDN

Given two nodes u and v in HDN(B, k, S), we first present a simple routing algorithm that finds a shortest path from u to v [6]. In Section II, we defined the product and quotient graphs. Now, we define the *difference graph* as follows. Let SN_1 and SN_2 are two super-nodes in base network B, the difference graph $SN_1 - SN_2$ is the sub-product graph of B such that $B_i, 1 \leq i \leq r$, is in $SN_1 - SN_2$ if and only if $B_i \subset SN_1$ and $B_i \not\subset SN_2$. For example, if $B = C_2 \times C_3 \times C_5$, $SN_1 = C_2 \times C_3$, and $SN_2 = C_3 \times C_5$ then $SN_1 - SN_2 = C_2$.

We also need a re-indexing process of nodes in the cluster, which is an HDN$(B, i-1, S)$, for routing via cross-edges of level i since the indexes of nodes in HDN$(B, i-1, S)$ is based on SN^{i-1} and the cross-edge of level i is defined based on SN^i. The index of a node in HDN$(B, i-1, S)$ contains four parts $(C^{i-1}, U_{id}^{i-1}, SN_{id}^{i-1}, N_{id}^{i-1})$.

At the construction of the ith level, HDN$(B, i-1, S)$ becomes a cluster containing only two parts, SN_{id}^i and N_{id}^i, of the node index in HDN(B, i, S). The other two parts, C^i and U_{id}^i, are generated from the construction at the ith level. The re-indexing process that generates an 1-to-1 mapping between $(C^{i-1}, U_{id}^{i-1}, SN_{id}^{i-1}, N_{id}^{i-1})$ and (SN_{id}^i, N_{id}^i) on an HDN$(B, i-1, S)$ is necessary for the proposed routing algorithm. More detailed explanation of re-indexing process is given in [6].

Assume that the point-to-point routing algorithm in the base network is available. The proposed algorithm for routing node u to node v in HDN(B, k, S) works as follows. We first perform re-indexing of u and v if $k > 1$. Then, there are three cases: the two nodes are in the same cluster (Case A), in the distinct clusters of distinct classes (Case B), and in the distinct clusters of the same class (Case C). Case A is trivial. Case C can be reduced to Case B by routing u via a cross-edge of level k. Therefore, we explain only the Case B: The two nodes are in the distinct clusters with the same class. We first identify the super-nodes, denoted as $SN_{u'}^k$ and $SN_{v'}^k$, in the two Q^{k-1}s containing u and v, respectively, such that $SN_{u'}^k$ and $SN_{v'}^k$ are connected by a unique cross-edge of level k in Q^k from the dual-construction. Then, we route node u to node u', and node v to node v' inside the clusters of level k, respectively. Notice that, u' and v' are not unique although $SN_{u'}^k$

and $SN^k_{v'}$ are unique. The algorithm finds the u' and v' that leave u^k_3 and v^k_3 unchanged if possible. And then, the routing from u to v is done by routing from u' to $u'' \in SN^k_{v'}$ via a cross-edge of level k in HDN(B, k, S) and routing from u'' to v' inside $SN^k_{v'}$. The algorithm is formally presented as Algorithm 1. The correctness of the algorithm and its time complexity are given in Theorem 2. Note that the Base_routing(B, u, v) in the algorithm is a routing algorithm in the base network (hypercube or torus for example).

Algorithm 1: HDN_routing(HDN$(B, k, S), u, v$)
  input: HDN(B, k, S);
  input: node $u = (u^k_0, u^k_1, u^k_2, u^k_3)$ (the node representation of level k);
  input: node $v = (v^k_0, v^k_1, v^k_2, v^k_3)$ (the node representation of level k);
  output: a path $u \Rightarrow v$;
begin
  if $k = 0$ **then**
    Base_routing(B, u, v);
  else
    if $k > 1$ **then** /* perform re-indexing */
      $(u^{k-1}_0, u^{k-1}_1, u^{k-1}_2, u^{k-1}_3) \leftarrow (u^k_2, u^k_3)$;
      $(v^{k-1}_0, v^{k-1}_1, v^{k-1}_2, v^{k-1}_3) \leftarrow (v^k_2, v^k_3)$;
    endif
    Case A: $u^k_0 = v^k_0$ and $u^k_1 = v^k_1$ /* u, v in the same cluster */
      if $k > 1$ **then**
        HDN_routing(HDN$(B, k-1, S), u, v$);
      else
        Base_routing(B, u, v);
      endif
    Case B: $u^k_0 \neq v^k_0$ /* u, v in the clusters of distinct classes */
      $u' \leftarrow (u^k_0, u^k_1, v^k_1, u^k_3)$;
      $v' \leftarrow (v^k_0, v^k_1, u^k_1, v^k_3)$;
      if $k > 1$ **then** /* perform re-indexing */
        $((u')^{k-1}_0, (u')^{k-1}_1, (u')^{k-1}_2, (u')^{k-1}_3) \leftarrow (v^k_1, u^k_3)$;
        $((v')^{k-1}_0, (v')^{k-1}_1, (v')^{k-1}_2, (v')^{k-1}_3) \leftarrow (u^k_1, v^k_3)$;
        HDN_routing(HDN$(B, k-1, S), u, u'$);
        HDN_routing(HDN$(B, k-1, S), v, v'$);
      else
        Base_routing(B, u, u');
        Base_routing(B, v, v');
      endif
      route u' to u'' via a cross-edge of level k; /* $u'' = (v^k_0, v^k_1, u^k_1, u^k_3)$ */
      Base_routing(B, u'', v'); /* route from u^k_3 to v^k_3 inside the super-node */
    Case C: $u^k_0 = v^k_0$ and $u^k_1 \neq v^k_1$ /* u, v in the clusters of the same class */
      route u to w via the cross-edge of level k;
      route node w to node v as in Case B;
  endif
end

Theorem 2 *Assume that the routing algorithm in the base network B is available. In HDN(B, k, S) for $k > 0$, routing between any two nodes can be done in at most $2^k R(B) - \sum_{j=0}^{k-1} 2^j R(SN^{k-j}) + 2^{k+1} - 2$ steps, where $R(B)$ and $R(SN^i), 1 \leq i \leq k$, are the time complexities of the routing in B and SN^i, respectively.*

Proof: We show the correctness of Algorithm 1 by induction on k. Assume that the algorithm is correct for $k - 1 \geq 0$. From the algorithm, it is clear that we need to consider only Case B. In Case B, nodes u' and u are in the same cluster by the definition of u'. They can be connected by the induction hypothesis. Similarly, nodes v' and v can be connected. The node u'' that is connected to u' by a cross-edge of level k and node v' are in the same super-node as can be seen from their IDs. Therefore, they can be connected by Base_routing algorithm. Next, we derive the time complexity R_k of the algorithm. In Case B, there are two recursive calls to connect u to u' and v to v', respectively. Since the node IDs of u and u' are the same (so are v and v'), a recursive call takes only $R_{k-1} - R(SN^k)$ time. Since the super-node IDs of u'' and v' are the same, the last call to Base_routing to connect u'' to v' takes only $R(SN^k)$ time. In Case C, there is an additional routing step via a cross-edge. Therefore, the time complexity R_k of HDN_routing(HDN(B, k, S), u, v) satisfies the recurrence $R_k = 2(R_{k-1} - R(SN^k)) + R(SN^k) + 2$ for $k > 0$. Solving this recurrence, we have

$$R_k = 2^k R(B) - \sum_{j=0}^{k-1} 2^j R(SN^{k-j}) + 2^{k+1} - 2$$

where $R(B)$ and $R(SN^i), 1 \leq i \leq k$, are the time complexities of the routing in B and SN^i, respectively. □

Lemma 1 *If two nodes are in distinct clusters of different classes, a path which connects the two nodes can go through only two clusters in which the two nodes reside.*

Proof: By the definition of the HDN, there is at least one cross-edge which connects the two clusters. The path can use this cross-edge(s) and route inside the clusters to reach the two nodes, respectively. □

Lemma 2 *If two nodes are in distinct clusters of the same class, a path connecting the two nodes can go through only three clusters.*

Proof: Two of them are the clusters in which the two nodes reside. The third cluster can be the one that has cross-edges connecting to the first two clusters and therefore it is of different class from that of the first two clusters. In the Case C of HDN_routing algorithm, we route node u to a node w in the third cluster via the cross-edge of level k. Instead, we can also route node v to a node w in the third cluster. After that, Lemma 1 can be applied. □

3.2 Distributing on HDN

To build multiple disjoint-paths between the nodes u and v on an HDN, the basic idea is to let the paths use as different clusters as possible so that they will be disjoint each others. Because we must find d disjoint-paths on an HDN which has a node degree of d, all the d neighbors of u or v will be used in the paths.

In this paper, we assume that the number of super-nodes (and hence the number of clusters of each class) is larger than the number of disjoint-paths. Some of neighbors may be located in the same super-node of u or v. In such case, we further route the neighbor to a node which is in an unused super-node. Algorithm 2, namely Base_distributing_any, performs the node *distributing* for a given node u in the base network B. The output of the algorithm is the d_0 paths, starting from u and ending at u_i or w_i, $0 \leq i < d_0$, each of which is located in a unique super-node.

Figure 4 shows an example of Algorithm 2. The base network is a $2 \times 3 \times 5$ Torus and a super-node contains 5 nodes (a 5-node ring). There are 6 super-nodes, namely SN_j, $0 \leq j < 6$, that are connected with a 2×3 Torus. The specified node u has 5 neighbors: u_i, $0 \leq i < 5$. Three of them

---

Algorithm 2: Base_distributing_any(B, s_1, u)
  input: B (base network);
  input: s_1 (the number of nodes in a super-node at level 1);
  input: node $u = (u_{SN}, u_N)$;
  output: distributed paths;
begin
  for $i = 1$ **to** d_0 **do**
    $u_i \leftarrow (u_{iSN}, u_{iN})$; /* u_i is a neighbor of u */
    if $u_{iSN} \neq u_{SN}$ and u_i's super-node is not in paths **then**
      path$[i] \leftarrow$ path$[i] \cup [u \rightarrow u_i]$;
    else
      find $w_i \leftarrow (w_{SN}, u_{iN})$ such that w_i's super-node is not in paths;
      newpath $\leftarrow$ Base_routing(B, u_i, w_i);
      /* newpath $= [u \rightarrow u_i \rightarrow ... \rightarrow w_i]$ */
      path$[i] \leftarrow$ path$[i] \cup$ newpath;
    endif
  endfor
end

(u_0, u_1, and u_2) are in different super-nodes (SN_0, SN_2, and SN_4). Two neighbors (u_3 and u_4) are in the same super-node of u. Thus we route them to nodes w_3 and w_4, which are located in SN_3, and SN_5, respectively.

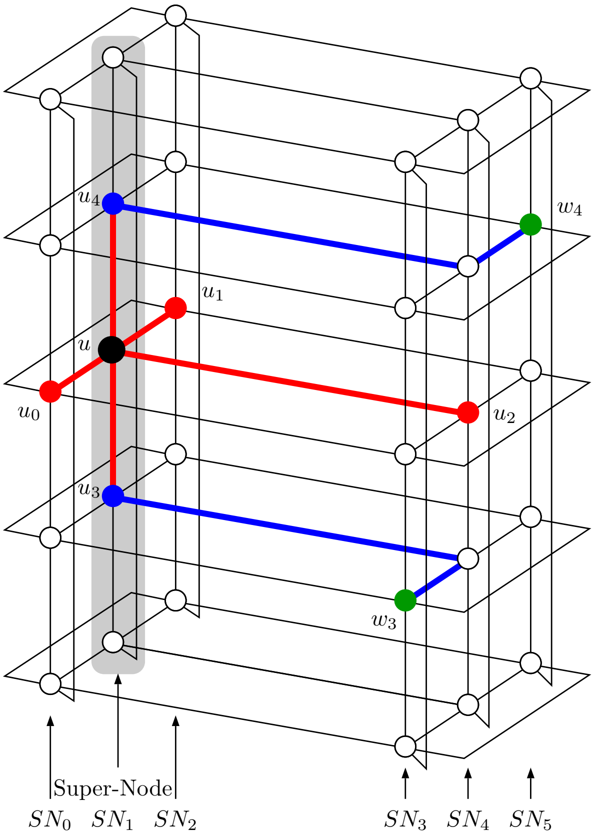


Figure 4: Distributing image on $2 \times 3 \times 5$ Torus

We call an ending node in the distributed path a *dispersion node*. Algorithm 2 finds the dispersion nodes inside the base network. Algorithm 3, as shown as below, finds the dispersion nodes in an HDN(B, k, S) with $k > 0$. The basic idea of the algorithm is to extend the distributed paths to reach clusters of different classes by using the cross-edges of level k.

Algorithm 3: HDN_distributing_any(HDN(B, k, S), u)
  input: HDN(B, k, S);
  input: node $u = (u_0, u_1, u_2, u_3)$;
  output: distributed paths;
begin
  if $k = 1$ **then** Base_distributing_any(B, s_1, u);
  else HDN_distributing_any(HDN$(B, k-1, S), u)$;
  endif
  for $i = 1$ **to** the number of paths **do**
    /* w_i is a dispersion-node connected by a cross-edge */
    $w_i \leftarrow (\overline{d_{i0}}, d_{i2}, d_{i1}, d_{i3})$;
    distributed_path$[i]$ ← path$[i] \cup w_i$;
  endfor
end

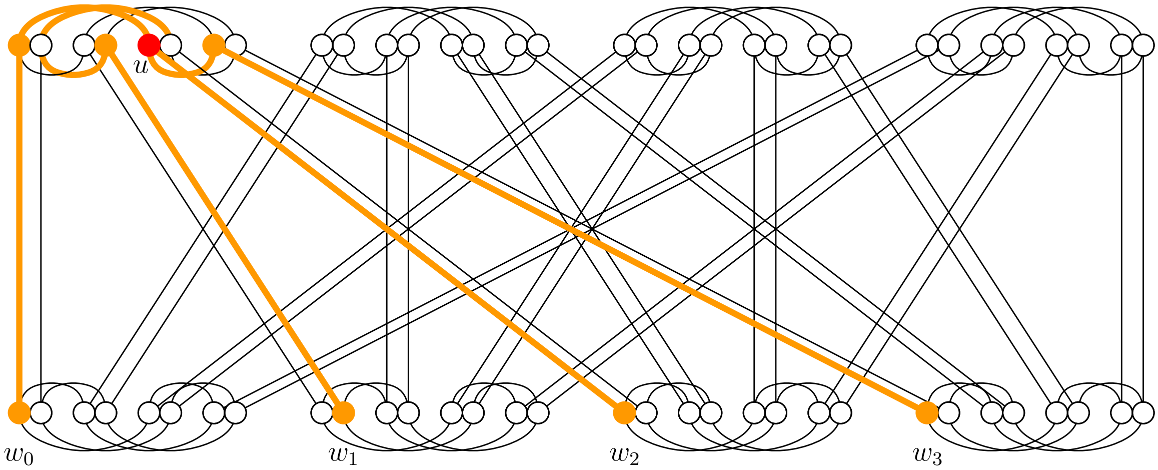


Figure 5: Distributing image on an HDN$(B, 1, S)$

As an example of Algorithm 3, Figure 5 shows finding distributed paths on an HDN$(B, 1, S)$ with a base network of a 3-cube. We let $s_1 = 2$. The node u is of class 0 and we must find $d_0 + k = 3 + 1 = 4$ distributed paths. First, the algorithm finds the $d_0 = 3$ distributed paths inside the base network. Each of the dispersion nodes after this step is located in a unique super-node. Then we extend the 3 paths to reach the clusters of class 1. Another path can be obtained by routing node u directly along with its cross-edge. Each of the 4 dispersion nodes after this step is located in a unique cluster of class 1.

Algorithm 2 and Algorithm 3 find distributed paths from node u to any dispersion nodes as long as the dispersion nodes are in different super-nodes or clusters. In the disjoint-path routing algorithm, it is needed to find a special distributed path from node u to a special dispersion node v. The following algorithms, Algorithm 4 and Algorithm 5, perform the node distributing with a pre-assigned dispersion node v in base network and HDN, respectively.

Algorithm 4 finds distributed paths of node u with a dispersion node v in the base network B. The algorithm determines this special path first, and then finds the rest paths by Algorithm 2.

Algorithm 4: Base_distributing(B, s_1, u, v)
  input: B (base network);
  input: s_1 (the number of nodes in a super-node at level 1);
  input: node $u = (u_{SN}, u_N)$;
  input: node $v = (v_{SN}, v_N)$;
  output: distributed paths, one of them ended at v;
begin
  if $\exists w(w_{SN} = v_{SN})$ /* w is neighbor of u */
    Base_routing(B, w, v);
  else
    $w' \leftarrow (w_{SN}, v_N)$;
    Base_routing(B, w, w');
    Base_routing(B, w', v);
  endif
  distribute the rest neighbors as Algorithm 2;
end

Figure 6 shows two examples of Algorithm 4. In Figure 6(a), a u's neighbor node w and the pre-assigned node v are in a same super-node, so we route node w to v inside the super-node. In Figure 6(b), we first route w to a w' whose node_id equals to the node_id of v inside the super-node and then route w' to v.

Similar to Algorithm 3, Algorithm 5 also distributes paths starting from node u on an HDN(B, k, S) but one of the path ends at a pre-assigned node v.

3.3 Disjoint-Path Routing on HDN

We have described the path distributing algorithms from a given node u. This subsection gives an algorithm for finding disjoint-paths on HDN. Suppose that a disjoint-path routing algorithm exists for a given symmetric base network B and we name it Base_disjoint_path(B, u, v).

Algorithm 6 is an algorithm for finding disjoint-paths on HDN. If $k = 0$, the algorithm calls Base_disjoint_path. For $k > 0$, the algorithm can be classified roughly three cases.

If the two nodes (u and v) are in the same cluster (Case A), $d_0 + k - 1$ paths are found recursively in the cluster. And, we find the rest path to use HDN_routing(u', v') where u' and v' are the neighbor nodes of u and v linked by the cross-edges, respectively.

If two nodes are in the distinct clusters of same class (Case C), we find distributed paths for u and v and connect the each pair of two dispersion nodes which are in same clusters. Since each node pair is in a separated cluster, the paths are disjoint.

If two nodes are in the distinct clusters of distinct classes (Case B), we first find distributed paths for u and v. One of the dispersion nodes, say w, in the distributed paths for node u may be located in the same cluster of v. In this case, we find distributed paths for v with a pre-assigned node w. If node v' (the neighbor of v linked with the cross-edge) is in the same cluster of u, then we find distributed paths for u with a pre-assigned node v'. After that, two dispersion nodes of u and v are connected by cross-edges.

Figures 7-9 illustrate disjoint-path routing examples on an HDN($B, 1, S$) with a 3-cube base network and $s_1 = 2$ by Algorithm 6. Figure 7 shows the case in which u and v are in a same cluster (Case A). Three disjoint-paths are built inside the cluster and one path is generated through cross-edges of u and v.

Figure 8 shows the disjoint-path routing example of Case B in which u and v are in distinct clusters of different classes. In Figure 8(a), the distributed paths of u were generated first. One

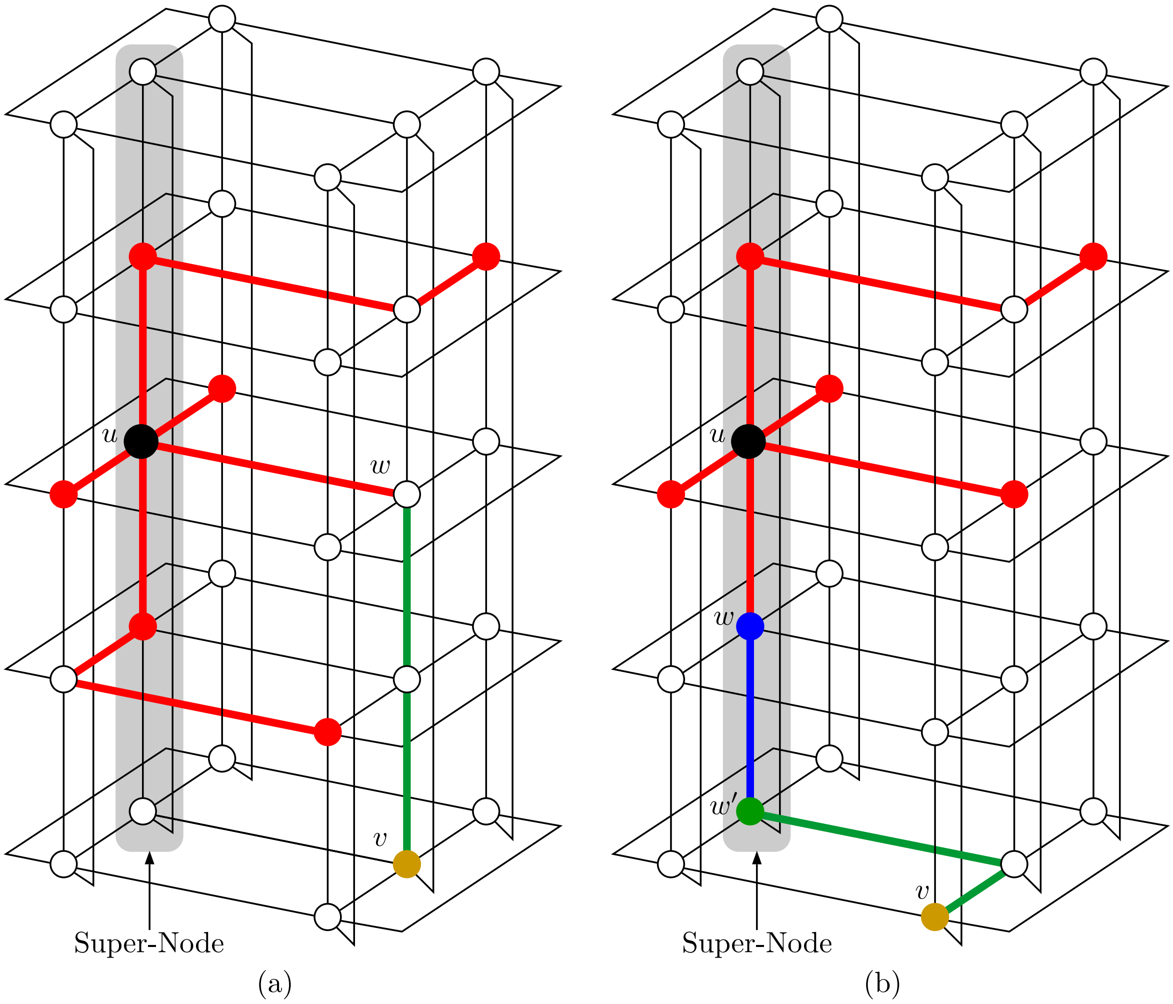


Figure 6: Distributing image with a pre-assigned dispersion node

dispersion node, w, is located in the cluster of v. Then the distributed paths were generated with the pre-assigned w. In Figure 8(b), because a neighbor of v, v', is in the cluster of u, the distributed paths of u were generated with the pre-assigned v'. As the result of node distributing, all the distributed paths of u or v are in different clusters. Then we can connect two nodes of each pair with cross-edges.

Figure 9 shows the disjoint-path routing example of Case C in which u and v are in distinct clusters of same class. This is a simple case. The distributed paths of u and v were generated and two dispersion nodes in the same cluster were connected.

Figure 10 shows the Case C in an HDN$(B, 1, S)$ with $k > 1$. We only show the clusters in the figure. Nodes u and v are located in the distinct clusters, Cluster$_u$ and Cluster$_v$, of a same class. The distributed paths of u and v were generated. All of the dispersion nodes are distributed to the clusters of another class. Then we can connect them through other clusters than Cluster$_u$ and Cluster$_v$.

Theorem 3 *If d_0 disjoint-paths in the base-network B exists, $d_0 + k$ disjoint-paths on HDN(B, k, S) can be found by Algorithm 6 if the number of clusters of each class is larger than or equal to $d_0 + k$.*

Proof: If two nodes are in the distinct clusters of same class (Case C), distributing two nodes and

Algorithm 5: HDN_distributing(HDN$(B,k,S),u,v$)
input: HDN(B,k,S);
input: node $u=(u_0,u_1,u_2,u_3)$;
input: node $v=(v_0,v_1,v_2,v_3)$;
output: distributed paths, one of them ended at v;
begin
  if $u_0=v_0$ and $u_1=v_1$ **then** /* u and v are in the same cluster */
    if $k>1$ **then** HDN_distributing(HDN$(B,k-1,S)$, u,v);
    else Base_distributing(B,u,v);
    endif
    for $i=1$ **to** number of dispersion-nodes **do**
      if $u_i\neq v$ **then** /* u_i is dispersion-node of u */
        $w\leftarrow u_i'$; /* w is connected to u_i by a cross-edge of level k */
        path$[i]$ $\leftarrow$ path$[i]\cup w$;
      endif
    endfor
  else
    HDN_distributing_any(HDN(B,k,S), u);
    if $\exists w(w_0=v_0$ and $w_1=v_1)$ **then** /* w is dispersion-node of u */
      HDN_routing(HDN(B,k,S), w,v); /* and is in cluster of v */
    else
      $w\leftarrow u_i'$; /* u' is an any dispersion-node of u */
      HDN_routing(HDN(B,k,S), w,v);
    endif
  endif
end

jointing each distributed-paths are done without problems. In the case that the two nodes are of distinct classes (Case B), if node u connects to a node u_k in the cluster in which v is contained, one of the distributed-path of v can be ended at u_k. If a neighbor of node v, v' in the cluster in which u is contained, one of the distributed-path of u can be ended at v'. If two nodes are in the same cluster (Case A), it is clear that d_0+k-1 disjoint-paths can be built inside the cluster without distributing all other clusters. The last path can be built outside of the cluster through the cross-edges of the two nodes. □

Theorem 3 gives a sufficient condition for the disjoint paths to exist. It is not the necessary condition. If the number of clusters of each class is less than d_0+k, the d_0+k disjoint-paths on HDN(B,k,S) may exist. To prove it and to develop a disjoint routing algorithm that eliminates the limitation on the cluster numbers are the future works. Theorem 4 gives the upper bounds on the maximum path length and time complexity of the current algorithm.

Theorem 4 *Algorithm 6 finds d_0+k disjoint paths between node u and node v of lengths at most $(3\times2^{k-1}+2)D(B)-3\sum_{j=0}^{k-2}2^jD(SN^{k-1-j})-D(SN^k)+3\times2^k+2k-2$ in $O(d_k2^k)$ time complexity.*

Proof: Figure 11 shows the case of the maximal path length (Case C) on HDN(B,k,S). In Base_distributing_any algorithm, the distance between node u and node u_i is 1. The distance between node u_i and w equals the diameter of graph B/SN^k. Therefore, the maximal length of path given by Base_distributing_any algorithm is $1+D(B)-D(SN^k)$. HDN_distributing_any

```
Algorithm 6: HDN_disjoint_path(HDN(B, k, S), u, v)
  input: HDN(B, k, S);
  input: node u = (u0, u1, u2, u3) (the node representation of level k);
  input: node v = (v0, v1, v2, v3) (the node representation of level k);
  output: disjoint-paths u ⇒ v;
begin
  if k = 0 then Base_disjoint_path(B, u, v);
  else
    Case A: u0 = v0 and u1 = v1                         /* u, v in the same cluster */
      if k > 1 then HDN_disjoint_path(HDN(B, k − 1, S), u, v);
      else Base_disjoint_path(B, u, v);
      w ← (u'0, u'1, w2, w3); /* u'2 ≠ w2 */
      /* u' is connected to u by a cross-edge of level k*/
      HDN_routing(HDN(B, k, S), u', w); /* route node u' to node w */
      /* v' is connected to v by a cross-edge of level k*/
      HDN_routing(HDN(B, k, S), w, v'); /* route node w to node v' */
    Case B: u0 ≠ v0                          /* u, v in the clusters of distinct classes */
      if v2 = u1 then                         /* v' and u are in the same cluster. */
        /* v' is connected to node v by a cross-edge of level k */
        HDN_distributing(HDN(B, k, S), u, v');                        /* Algorithm 5 */
      else HDN_distributing_any(HDN(B, k, S), u);                     /* Algorithm 3 */
      if ∃w(w0 = v0 and w1 = v1) then                 /* w is dispersion-node of u */
        HDN_distributing(HDN(B, k, S), v, w);                /* w and v: same cluster */
      else HDN_distributing_any(HDN(B, k, S), v);                     /* Algorithm 3 */
      for i = 1 to number of distributed-path do
        ui ← (ui0, ui1, ui2, ui3);                 /* ui is ith rest dispersion-node of u */
        vi ← (vi0, vi1, vi2, vi3);                 /* vi is ith rest dispersion-node of v */
        HDN_routing(HDN(B, k, S), ui, vi);                  /* route node ui to node vi */
    Case C: u0 = v0 and u1 ≠ v1               /* u, v in the clusters of the same class */
      HDN_distributing_any(HDN(B, k, S), u);                          /* Algorithm 3 */
      HDN_distributing_any(HDN(B, k, S), v);                          /* Algorithm 3 */
      for i = 1 to number of cluster which contain two dispersion-nodes do
        ui ← (ui0, ui1, ui2, ui3);            /* ui is dispersion-node which is in the cluster */
        vi ← (vi0, vi1, vi2, vi3);            /* vi is dispersion-node which is in the cluster */
        HDN_routing(HDN(B, k, S), ui, vi);    /* route node ui to node vi in the same cluster */
      for i = 1 to number of the rest of dispersion-node of u do
        ui ← (ui0, ui1, ui2, ui3); /* ui is ith rest dispersion-node of u */
        vi ← (vi0, vi1, vi2, vi3); /* vi is ith rest dispersion-node of v */
        HDN_routing(HDN(B, k, S), ui, vi); /* route node ui to node vi */
  endif
end
```

algorithm connects dispersion-node and node w_i on each level. Therefore, the maximal length of path is $D(\text{Dist}) + 1 = 1 + D(B) - D(SN^k) + k = D(B) - D(SN^k) + k + 1$. The maximum length between one dispersion-node u_i and a node v_i whose super-node_id is same as u_i is

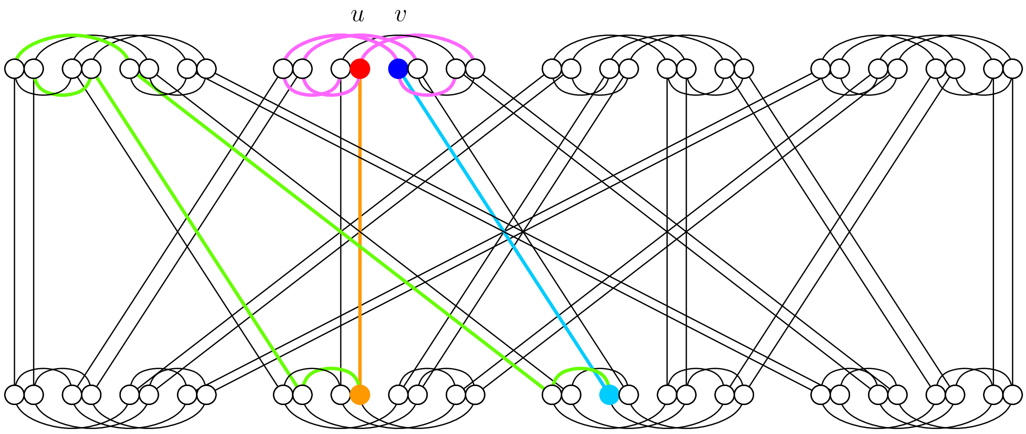
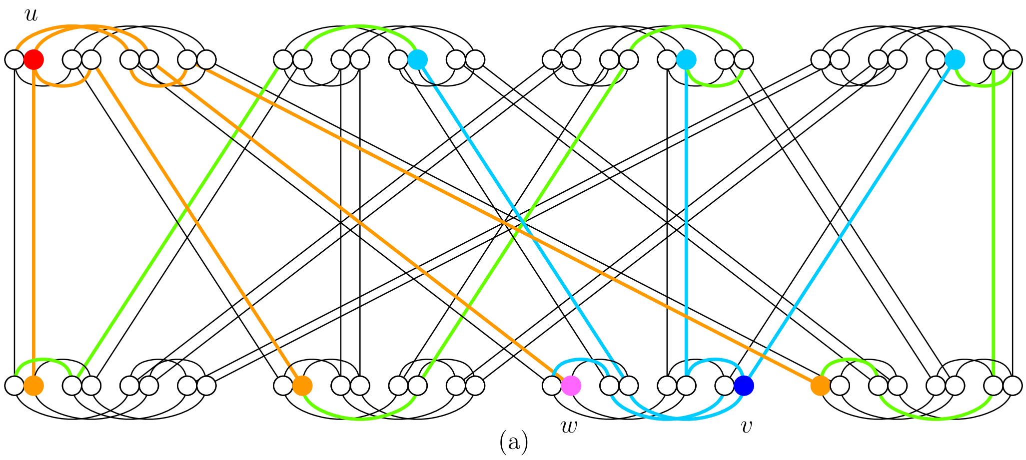
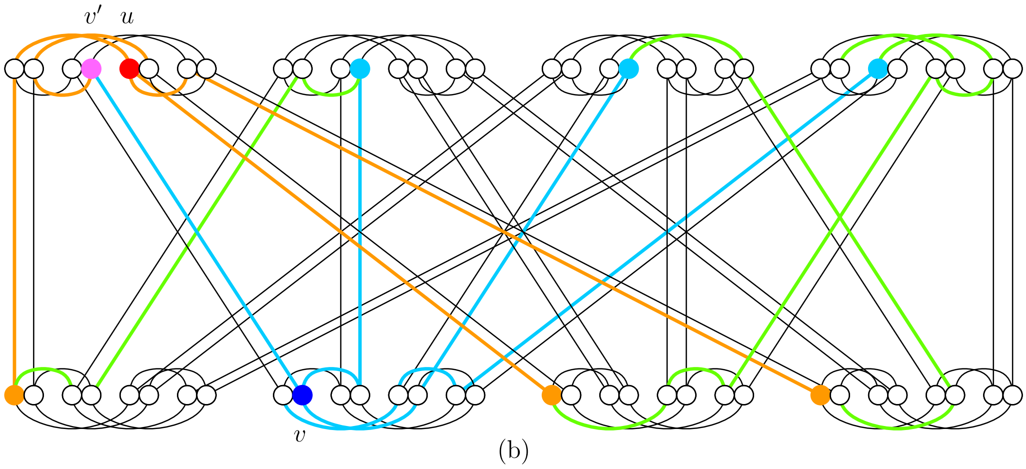


Figure 7: Case A: disjoint-path image on HDN$(B, 1, S)$



(a)



(b)

Figure 8: Case B: disjoint-path image on HDN$(B, 1, S)$

$(D(Q^{k-1})+1)+(D(Q^{k-1})+1)+D(Q^{k-1}) = 3D(Q^{k-1})+2$. Moreover, a $D(SN^k)$ length is needed to route to node_id in a super-node. From the above, the maximal length is $2(D(B)-D(SN^k)+k+1)+(3D(Q^{k-1})+2)+D(SN^k) = 3D(Q^{k-1})+2D(B)-D(SN^k)+2k+4$. Solving this recurrence,

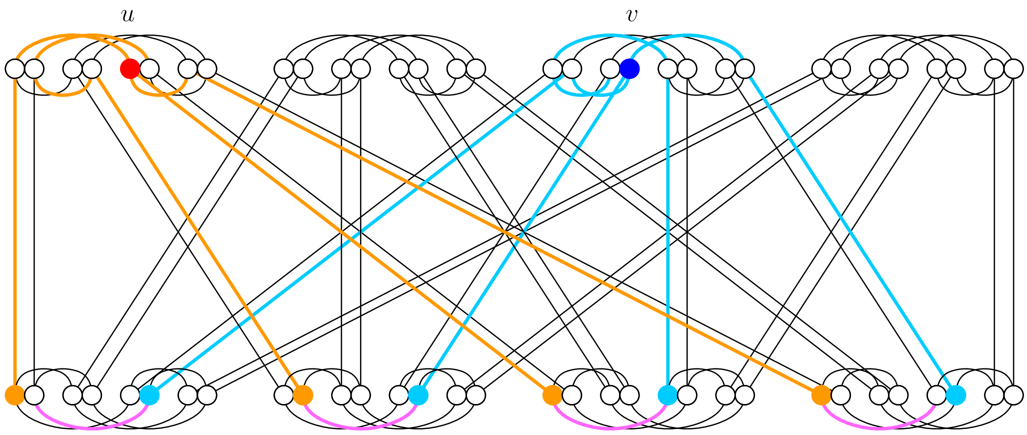


Figure 9: Case C: disjoint-path image on HDN$(B, 1, S)$

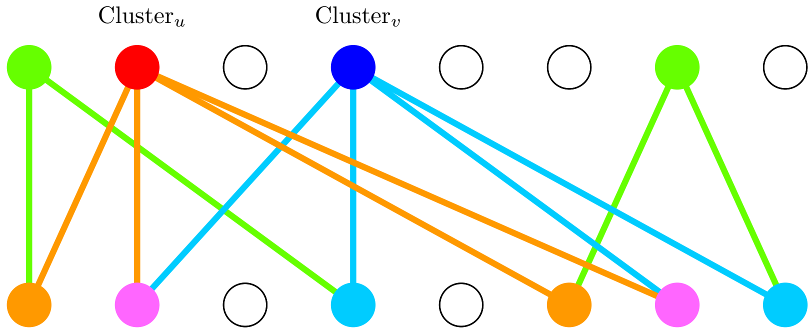


Figure 10: Case C: disjoint-path image on HDN(B, k, S) with $k > 1$

we get $(3\times 2^{k-1}+2)D(B)-3\sum_{j=0}^{k-2}2^j D(SN^{k-1-j})-D(SN^k)+3\times 2^k+2k-2$. The time complexity of Figure 11 is most high. In this case, Base_routing is called 2^k times, so the time complexity of Algorithm 6 is $O(d_k 2^k)$. □

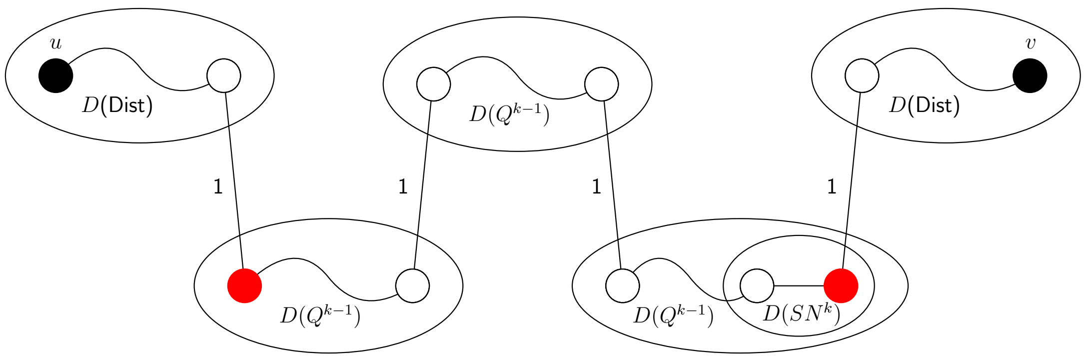


Figure 11: The case of maximal path length on HDN(B, k, S)

If some nodes in HDN are faulty, all the $d_0 + k$ may not be found by the algorithm. If the algorithm can find at least one path between two nodes, then the communication between the two nodes can be done. We give the simulation results by considering the node faulty in the next section.

4 Experimental Results

We have performed a set of simulations on the performance of the proposed algorithms with faulty nodes. Our simulations focused on 1) the successful rate of finding all the disjoint-paths, 2) the successful routing rate, i.e., there is at least one path connecting the two nodes, and 3) the average path lengths, for an HDN with different node faulty probabilities.

We used a 3-cube as the base network in which three disjoint-paths exist. We let $k = 2$ and $S = \{2, 8\}$. Therefore, the numbers of nodes in HDN$(B, 1, S)$ and HDN$(B, 2, S)$ are $8 \times 8/2 \times 2 = 64$ and $64 \times 64/8 \times 2 = 1024$, respectively. The diameters of the HDN$(B, 1, S)$ and HDN$(B, 2, S)$ are $(1 + 3 - 1) \times 2 + 1 = 7$ and $(1 + 7 - 3) \times 2 + 3 = 13$, respectively. The number of disjoint-paths is $d_0 + k = 3 + 2 = 5$. The simulation consists of the following four steps.

1. Mark faulty nodes in HDN$(B, 2, S)$ randomly at a specified percentage of the faulty nodes.

2. Select two non-faulty nodes, u and v, in HDN$(B, 2, S)$ randomly.

3. Find disjoint-paths for the two nodes by using the HDN_disjoint_path algorithm.

4. Record the the number of successful paths, the number of path lengths, and so on.

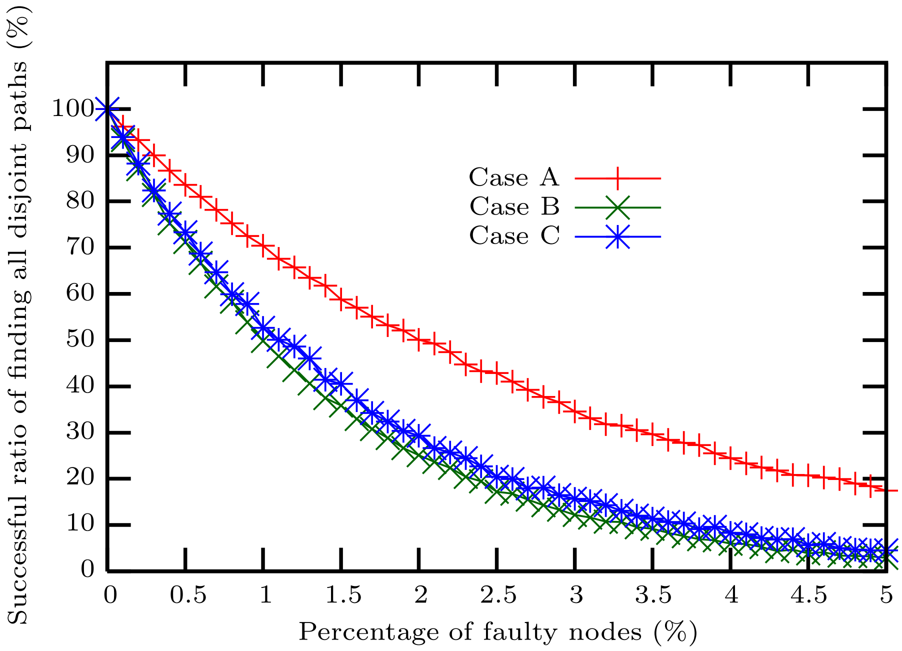


Figure 12: Successful ratio of finding all disjoint-paths

Figure 12 illustrates the successful ratio of finding all disjoint-paths. The x-axis of the graph is faulty rate and the y-axis is successful ratio of finding all disjoint-paths. The probability of finding all disjoint-paths ($P_{all}(F, L_{sum})$) is expressed the following expression.

$$P_{all}(F, L_{sum}) = (1 - F)^{L_{sum}} \tag{1}$$

where F is the probability of the node faulty and L_{sum} is number of nodes in all the disjoint-paths.

Figure 13 illustrates the successful communication rate. If there is a path connecting the two nodes, we say they can communicate successfully. From the figure, we can see that the Case A has highest rate than other cases.

Figure 14 shows the average path lengths. The lengths become shorter as the faulty rate increases. This is because, as the faulty rate increases, the successful routing rates decreases, especially for

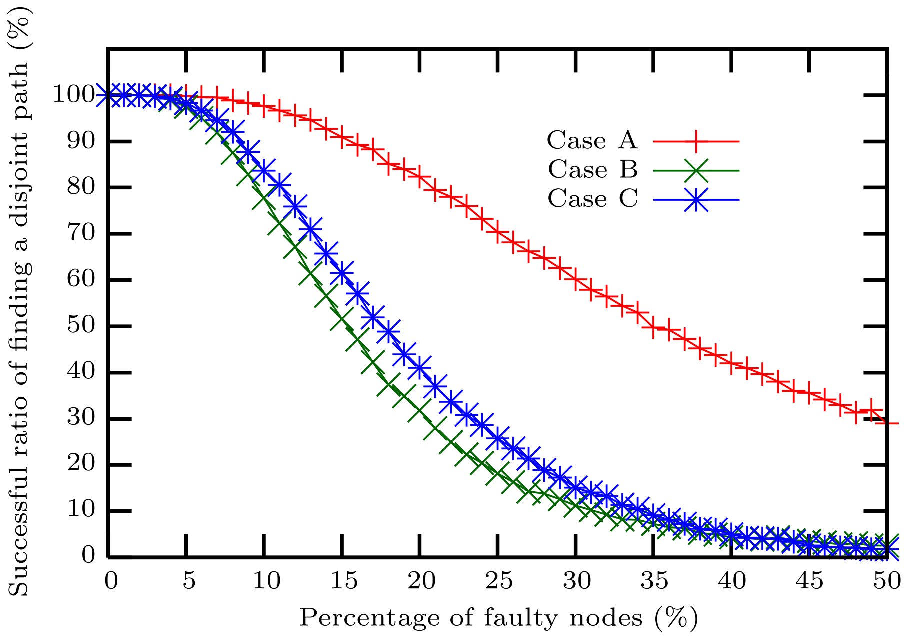


Figure 13: Successful ratio of finding a path

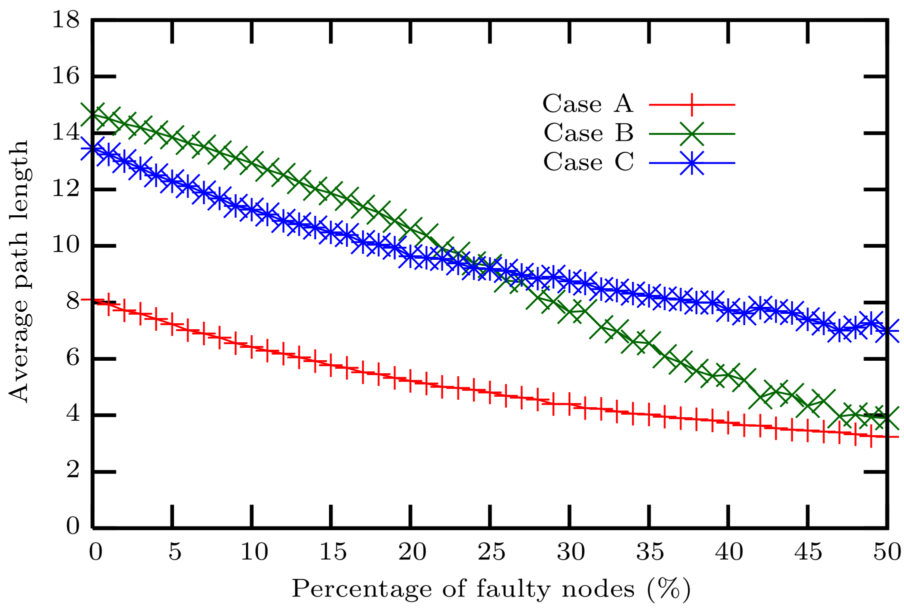


Figure 14: Average path length

those paths which have longer lengths, and we just count the paths which successfully connect the two nodes. Again, the Case A has shortest path lengths than other cases.

As the number of clusters the disjoint-path uses increases, the path length is tend to be longer. This is because that the path needs more cross-edges to move to distinct clusters. We have examined the number of clusters the disjoint-path contains for each case.

In Case A, four paths which are generated in the cluster use only one cluster. If two specified nodes u and v are in the same super-node, then the rest paths use two clusters. Otherwise the paths use four clusters. In Case B, if the super-node_id of the one specified node is equal to the cluster_id of the other node, then a path uses two clusters and the rest paths use four clusters. Otherwise all the paths use four clusters. The paths use three or five clusters in Case C. If a path has two dispersion-nodes which are in a same cluster, the path uses three clusters. Otherwise it uses five clusters.

Based on the discussion above, we calculated the expected numbers of clusters the disjoint paths used for the three cases. E_{caseA} is the expected average number of clusters in case A. Four paths are generated inside the cluster. Another path is generated outside the cluster. In our simulation, there are eight clusters of each class. The probability the two nodes are in a same super-node is 1/8. In this case, the path can be generated by using another cluster. That is, two clusters are used. Otherwise, four clusters will be used. Therefore we get the expected value of E_{caseA} as below:

$$E_{caseA} = (4 \times 1 + 1/8 \times 2 + 7/8 \times 4)/5 = 1.55$$

E_{caseB} is the expected average number of clusters in case B. The probability the super-node_id of a node equals the cluster_id of the other node is $1/8 + 7/8 \times 1/8 = 15/64$. Therefore we get the expected value of E_{caseB} as below:

$$E_{caseB} = (4 \times 4 + (15/64 \times 2 + 49/64 \times 4))/5 = 3.90$$

In Case C, the expected number of clusters in which two dispersion-nodes exist is 25/8. Therefore we get the expected value of E_{caseC} as below:

$$E_{caseC} = (25/8)/5 \times 3 + (5 - 25/8)/8 \times 5 = 3.75$$

The expected numbers of clusters the disjoint-paths use the in three cases are calculated at the assumption that there is no faulty node. We get $E_{caseB} > E_{caseC} > E_{caseA}$. This is consistent with the simulation results shown in Figure 14. However, as the number of faulty nodes increases, the number of paths we can find decreases. The E_{caseB} and E_{caseC} converge 2 and 3, respectively, thus $E_{caseB} < E_{caseC}$. This is also consistent with the simulation results. From Figure 14, we found that the relation between E_{caseB} and E_{caseC} is reversed when the faulty rate is around the 23%.

We did not take any measures for fault-tolerant routings in this paper. If the fault-tolerant routing is taken into account, better successful rates of finding all disjoint-paths and the communications are expected.

5 Concluding Remarks

In this paper we showed that there are $d_0 + k$ disjoint-paths on HDN(B, k, S) if there are d_0 disjoint-paths on the base network B and the number of clusters of each class is larger than $d_0 + k$. Moreover, we proposed an algorithm to find the disjoint-paths on HDN(B, k, S) and investigated the performance of the algorithm via simulations. The future work may include the fault-tolerant routing and eliminating the limitation on the cluster numbers.

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*, 49(2/3):265–276, 2005.

[2] K. Ghose and K. R. Desai. Hierarchical cubic networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):427–435, April 1995.

[3] IBM. *Roadrunner: Hardware and Software Overview.* IBM Corporation, http://www.redbooks.ibm.com/redpapers /pdfs/redp4477.pdf, 2009.

[4] Y. Li, S. Peng, and W. Chu. Efficient collective communications in dual-cube. *The Journal of Supercomputing*, 28(1):71–90, April 2004.

[5] Y. Li, S. Peng, and W. Chu. Recursive dual-net: A new universal network for supercomputers of the next generation. In *Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing*, pages 809–820, Taipei, Taiwan, June 2009.

[6] Y. Li, S.Peng, and W. Chu. Hierarchical Dual-Net: A Flexible Interconnection Network and its Routing Algorithm. *International Journal of Networking and Computing*, 2(2):234–250, 2012.

[7] F. P. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, 24:300–309, May 1981.

[8] Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7):867–872, July 1988.

[9] TOP500. *Supercomputer Sites.* http://top500.org/, Nov. 2013.