Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 12, Number 1, pages 22–30, January 2023

Hybrid interconnection topologies for high performance and low hardware cost based on hypercube and k-ary n-tree

Junhong Li
Department of Computer Science
Hosei University
junhong.li.2k@stu.hosei.ac.jp

Yaodong Wang
Department of Computer Science
Hosei University
yaodong.wang.7y@stu.hosei.ac.jp

Yamin Li
Department of Computer Science
Hosei University
yamin@hosei.ac.jp

***Abstract*—In the field of high-performance parallel computing, interconnection networks based on fat-tree topology have been widely used. A leaf-level switch can be connected to k computing nodes through k links for a traditional fat-tree structure. If a large-scale high-performance also uses traditional fat-tree, it requires a large number of switches and links to connect computing nodes, which will significantly increase the hardware cost. In this regard, this paper proposes two hybrid topologies to solve this problem by combining fat-tree and hypercube, named k-ary n-tree k-cube (KANTC) and Mirrored k-ary n-tree k-cube (MiKANTC). Instead of connecting k compute nodes to a leaf switch directly, we replace all the leaf-level switches of the k-ary n-tree with k-cubes. In this way, the leaf level will have k^{n-2} k-cubes, where each cube will select k switches to connect to the upper level of the k-ary n-tree, and the rest of the switches will be used to connect to the compute nodes. Each cube can connect $k(2^k - k)$ compute nodes. We give routing algorithms based on shortest path, and evaluated path diversity, cost, performance for KANTC and MiKANTC. The results show that the KANTC and MiKANTC can save 84% of switches and 78% of links in massively parallel systems when $k = n = 8$, compared to fat trees, and both KANTC and MiKANTC have higher path diversity than fat-tree.**

***Index Terms*—interconnection network, fat-tree, hypercube, hardware cost, routing algorithm, path diversity**

I. Introduction

In high-performance computing systems, large-scale interconnection networks are essential. Nowadays, with the development of distributed computing and cloud computing technologies, the number of compute nodes integrated into high-performance computing systems is increasing [1], and the size of interconnection networks is getting larger. Therefore, the trade-off between the cost and performance of large-scale interconnection networks becomes a critical issue for high-performance computing. Various interconnection networks have been designed to achieve a better trade-off between performance and hardware cost. Among the top500 supercomputers, the fat-tree [2] is one of the most commonly used interconnect topologies. The supercomputer Summit [3] (ranked 4th in the top500) used a fat-tree network based on InfiniBand interconnects. The fat-tree network isolates traffic between compute partitions and storage subsystems, providing a more predictable application performance. In addition, the high redundancy of this network and its reconfigurability ensure reliable high performance even after the failure of network components [4]. In a traditional tree network topology, the bandwidth is converged level by level, and the network bandwidth at the tree's root is much smaller than the sum of all bandwidths at the separate leaves.

On the other hand, a fat-tree is more like a real tree, where the branches get thicker the further it gets to the root, i.e., the network bandwidth does not converge from the leaves to the root. It is the basis for fat trees to be able to support non-blocking networks. However, the scalability of traditional fat-tree is theoretically limited by the number of ports in the core layer switches, which is not conducive to the long-term development requirements of data centers. To make fat-tree more scalable, some fat-tree schemes with multiple roots have been proposed, such as the k-ary n-tree proposed by Petrini and Vanneschi [5]; where k both represents the number of links to the upper or lower layers, and the number of compute nodes connected to one leaf switch. The n is the number of layers, which means that the structure can always maintain a fixed number of switch ports, regardless of the size. It is very conducive to topology scaling.

Gómez et al. proposed a reduced unidirectional fat-tree (RUFT) [6], [7], a structure that reduces the hardware cost of the switches. All packets must traverse from the first level switch to the last level switch and then traverse along the long link to the compute node because the links in RUFT are unidirectional. Ludovici et al. showed that RUFT is a more powerful option than traditional butter in implementing network-on-chip (NoC) [8]. Wang et al. pointed out that due to the complexity of the topology, the floor plan design of fat-tree-based NoC is very challenging and proposed a method to optimize the fat-tree floor plan that can effectively reduce the number of intersection points and minimize the interconnect length [9]. Li proposed the Mirror k-ary n-tree (MiKANT) network [10] to reduce the hardware cost of fat-tree by connecting more compute nodes with fewer switches and links, thus reducing the hardware cost and making it easier to be implemented than fat-tree and bidirectional Clos networks. Also, MiKANT shortens the average distance to reduce communication time and achieve high performance. Wang gave the link fault-tolerant routing algorithm in the MiKANT interconnection network [11] and evaluated its performance by simulation.

The hypercube [12] structure is widely used due to its elegant topological properties and ability to simulate various other commonly used networks. However, hypercubes have a significant drawback: the number of communication links per node is a logarithmic function of the number of nodes in the network. It makes traditional hypercube networks unsuitable for large-scale scaling. In [13], Arai and Li proposed a variant topology based on the hypercube called the Generalized-Star cube (GSC) and gave its routing algorithm. Wang proposed a hybrid structure based on hypercube and fat-tree [14], and presented a method for evaluating path diversity.

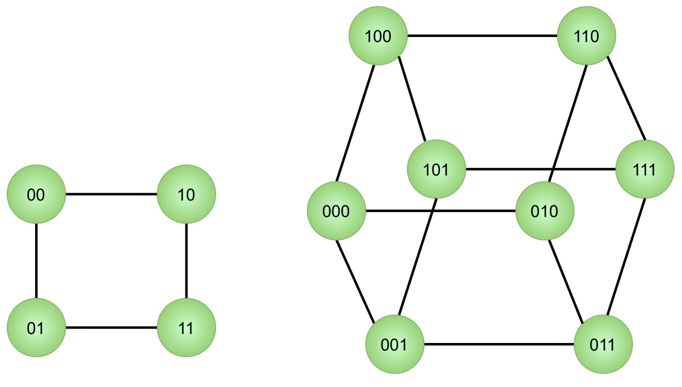
To solve these problems, we propose two hybrid topologies: k-ary n-tree k-cube (KANTC) and Mirrored k-ary n-tree k-cube (MiKANTC) based on fat trees and n-dimensional hypercubes. For KANTC/MiKANTC, the parameter k represents the dimensionality of the hypercube and the switch arity to ensure that the radix of the architecture is equal to 2k. Instead of connecting k compute nodes to the leaf switches directly, we replace all the leaf-level switches of the k-ary n-tree with hypercubes. Thus, the leaf level will have k^{n-2} cubes, each cube will choose k switches to connect to the upper level of the k-ary n-tree, and the rest of the switches will be used to connect to the computational nodes. Each cube can connect $k(2^k - k)$ compute nodes. We first calculate the number of compute nodes, links, and switches, the diameters of KANTC and MiKANTC. Then, we analyze the hardware cost, performance, and path diversity of KANTC and MiKANTC. The results show that KANTC and MiKANTC achieve higher path diversity with lower hardware costs than the traditional fat-tree.

The rest of the paper is organized as follows. Section II briefly describes the related work. Section III describes the topologies of KANTC and MiKANTC and their topological properties. Section IV focus on routing algorithms. Section V evaluates the performance of KANTC and MiKANTC. Section VI concludes the paper.

II. Related Work

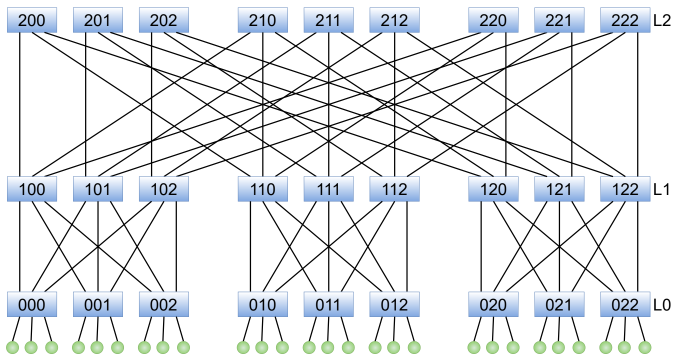
A. *n-dimensional Hypercube*

An n-dimensional hypercube, or n-cube for short, is a commonly used interconnected network with 2^n nodes and $n2^{n-1}$ links, and its node degree and diameter are n. A binary sequence of dimensions can represent any switch node. For any two nodes, there are links between them when and only when their binary code sequence differ by only one bit. Fig. 1 1(a) and 1(b) show a 2-cube and a 3-cube, respectively.



(a) 2-cube (b) 3-cube

Fig. 1. n-Dimensional Hypercube


200 201 202 210 211 212 220 221 222 L2
100 101 102 110 111 112 120 121 122 L1
000 001 002 010 011 012 020 021 022 L0
Fig. 2. A 3-ary 3-tree

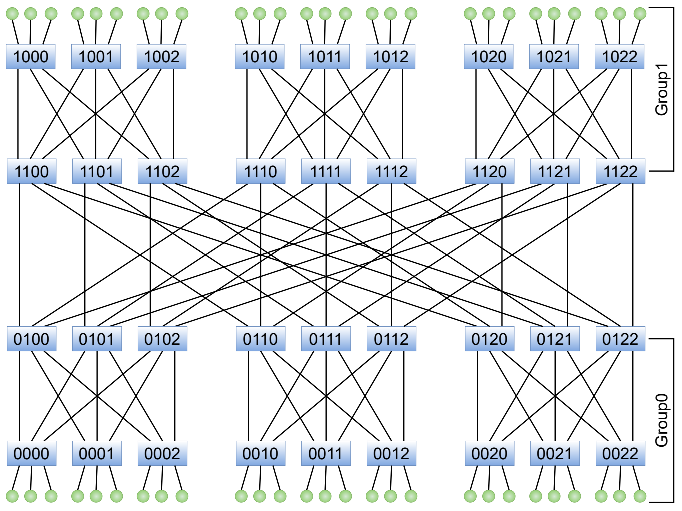


Fig. 3. A Mirrored 3-ary 3-tree

B. *k-ary n-tree and Mirrored k-ary n-tree*

A k-ary n-tree is a special case of a fat-tree, and a Mirrored k-ary n-tree, or MiKANT for short, is a special case of a k-ary n-tree. k-ary n-tree has k^n compute nodes, nk^{n-1} switches, and nk^n links. Each switch in level 0 is connected to k compute nodes. Fig. 2 shows a 3-ary 3-tree, where rectangles represent switches and circles represent compute nodes. There are $k^n = 27$ compute nodes, $nk^{n-1} = 27$ switches, and $nk^n = 81$ links. Each switch of the k-ary n-tree is labeled as $\langle L, D \rangle$, where L(Level) denotes the stage and $L \in \{0, .., n-1\}$, and $D = D_{n-2}, D_{n-3}, ...D_1, D_0$ is a $(n-1)$-tuple $\{0, 1, ..., k-1\}^{n-1}$, which identifies the switches of stage L. MiKANT has $2n-2$ stages; each stage has $k-1$ switches. The upper $n-1$ stages are symmetric to the lower $n-1$ stages. The fractional number of all switches is $2k$. the diameter of MiKANT(k,n) is $2n$. the partition width of MiKANT(k, n) is $k/2$. In MiKANT(k,n), each switch is labeled as $\langle G,L,D\rangle$ where G (group) denotes the group ID with $G\in 0,1$, and L and D are the same as in the k-ary n-tree. Fig. 3 shows a MiKANT (3,3). There are $(2n-2)k^{n-1}=36$ switches, $2k^n=54$ compute nodes, and $(2n-1)k^n=135$ links.

III. KANTC and MiKANTC

This section describes how KANTC and MiKANTC are encoded and their topological characteristics.

A. k-ary n-tree k-cube

k-ary n-tree k-cube is motivated by the k-ary n-tree and n-dimensional hypercube. It is represented as KANTC(k,n), where k indicates both the dimension of hypercube and the arity of the k-ary n-tree. KANTC(k,n) is divided into two parts, the upper part consists of $n-1$ layers of the k-ary n-tree, and the lower part consists of k^{n-2}cubes. In other words, we replace all the leaf-level switches of the k-ary n-tree with k-cubes instead of directly connecting the k compute nodes to the leaf switches. Each k-cube can connect $k(2^k-k)$ compute nodes. In a hypercube of KANTC, k switches connect to the upper k-ary n-tree. These k switches are also part of the k-ary n-tree, and we call these switches intermediate switches and the remaining 2^k-k switches are used to connect computational nodes.

For the encoding of KANTC(k,n), we propose a hybrid encoding approach based on k-ary n-tree and hypercube. Each switch is marked as $\langle L,D,C\rangle$, as shown in the fig. 4, where C represents the k-bit binary encoding of a hypercube switch, a $C=C_{k-1},C_{k-2},...,C_1,C_0$, where $C_i\in\{0,1\}$. As mentioned before, k intermediate switches need to be selected in a hypercube. We have chosen a bitwise inverse approach to ensure that the intermediate switches are distributed as evenly as possible. First, it needs to determine the encoding of the $\lceil k/2\rceil$ switches, picking all zeros as the first switch, and For switch i starting from the second switch, we invert all the values from bit i to bit $k-i$of the previous switch until the $\lceil k/2\rceil$th switch. For the remaining $\lfloor k/2\rfloor$ switches, we inverted all of the previously decided switches until the $\lfloor k/2\rfloor$ switch. For example, for a four-dimensional hypercube, we choose $\langle 0,0,0,0\rangle$, $\langle 0,1,1,0\rangle$, $\langle 1,0,0,1\rangle$, $\langle 1,0,0,1\rangle$ $\langle 1,1,1,1\rangle$ as intermediate switches. the $\langle L,D\rangle$ part of the KANTC code, consistent with the k-ary n-tree,where the $\langle C\rangle$ part of the n-1 to 1 layer will remain all-zero, and the 0th layer is also used as part of the hypercube, $\langle C\rangle$ is consistent with the hypercube, and for the convenience of the routing, the group will be aligned with its group of the k-ary n-tree in which it is located.

Fig. 4 shows a KANTC($3,4$). There are $k^{n-2}=9$ 3-cubes, and each hypercube is connected to the above tree through the intermediate switch. In a 3-cube, there are $2^k=8$ switches (including the intermediate switch). In addition to the intermediate switches, each 3-cube switch connects three compute nodes. There are 135 compute nodes.

B. Mirrored k-ary n-tree k-cube

Mirrored k-ary n-tree k-cube, represented as MiKANTC (k,n), is assembled based on MiKANT(k,n) and k-cube. Similar to the encoding of KANTC, each switch of MiKANTC(k,n) is marked as $\langle G,L,D,C\rangle$, where G represents a group; L represents a level; D is the switch ID in the L-level of the group G, and C is the ID in the k-cube. For example, a MiKANTC($3,4$) has $2k^{n-2}=18$ 3-cubes. There are 270 compute nodes. The level 2 switches of group 0 and group 1 form a KANTC($3,4$); the level 2 switches of group 1 and group 0 form another KANTC($3,4$). If $0\le level<n-2$, a switch W

$$\langle G,L,D_{n-2},...,D_{L+1},D_L,D_{L-1},...,D_0,C_{k-1},...,C_0\rangle$$

Connect to switch

$$\langle G,L+1,D_{n-2},...,D_{L+1},*',D_{L-1},...,D_0,C_{k-1},...,C_0\rangle$$

Otherwise,$L=n-2$ when connected to the switch

$$\left\langle\overline{G},L,*',D_{n-3},...,D_1,D_0,C_{k-1},...,C_0\right\rangle$$

where $\overline{G}$ is a bit transition of G and $*'$ is any value of $*'\in\{0,1,...,k-1\}$. For example, in MiKANTC($3,4$), a switch W$\langle 0,2,1,1,0,0,0,0,0\rangle$ is connected to switches $\langle 1,2,0,1,0,0,0,0\rangle$, $\langle 1,2,1,1,0,0,0,0\rangle$, and $\langle 1,2,2,1,0,0,0,0\rangle$ in a different group. When the source and destination nodes are in the same group, MiKANTC acts the same as KANTC.

C. Topological Properties of KANTC

Theorem 1. *The radix of the switch of KANTC(k, n) is* $2k$.

Proof. KANTC(k,n) can be divided into upper and lower parts, the upper part consists of $n-1$ layers switches of a k-ary n-tree, and the lower part consists of k^{n-2} cubes. The switch radix of k-ary n-tree is $2k$. In the k-cube part, each intermediate switch is connected to k switches of a k-ary n-tree and k other k-cube switches, and each switch connected to k compute node is connected to other k switches. The switch radix of the k-cube is also $2k$. Therefore, the radix of the switch of KANTC(k,n) is $2k$. □

Theorem 2. *There are* $(2^k-k)k^{n-1}$ *computing nodes in KANTC(*k,n*).*

Proof. KANTC(k,n) only the lower half of the cube switches are used to connect the compute nodes. There are k^{n-2} k-cubes in total, and there are 2^k-k switches connecting computing nodes in a k-cube; each switch connects k computing nodes. That is, the total number of computing nodes in KANTC(k, n) is $k^{n-2}\times(2^k-k)\times k=(2^k-k)k^{n-1}$. □

Theorem 3. *There are* $(n-1)k^{n-1}+2^kk^{n-2}$ *switches in KANTC(*k,n*).*

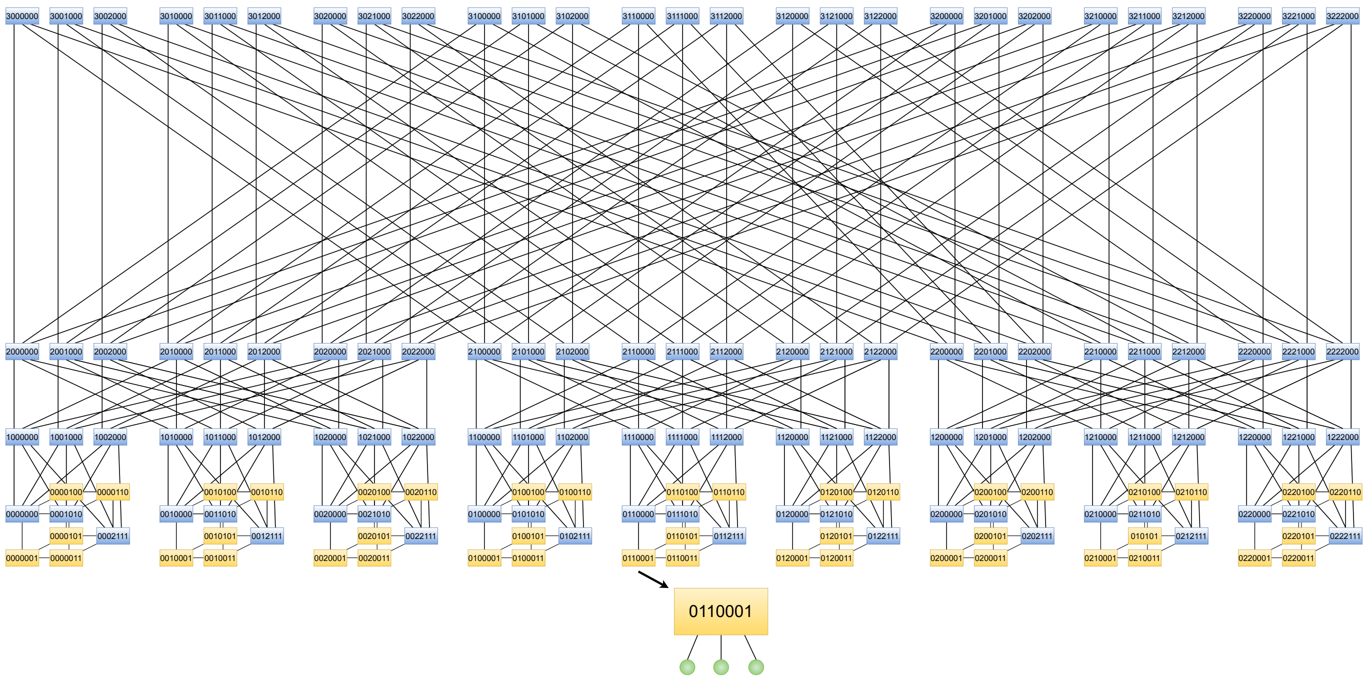


Fig. 4. A 3-ary 4-tree 3-cube

Proof. In the upper half of KANTC(k, n), there are $n-1$ layers, each layer has k^{n-1} switches, and in the lower half, there are k^{n-2} k-cubes, and each cube has 2^k switches. $(n-1) \times k^{n-1} + 2^k \times k^{n-2} = (n-1)k^{n-1} + 2^k k^{n-2}$ switches. □

Theorem 4. *There are $(n-1)k^n + (2^{k-1} + 2^k - k)k^{n-1}$ links in KATC(k, n).*

Proof. The number of links includes links between switches and links connecting compute nodes and switches. In the upper half of KANTC(k, n), there are $n-1$ layers, each layer has k^{n-1} switches, each switch provides k links, and in the lower half, there are k^{n-2} k-cubes, each k-cube has $k2^{k-1}$ links, and finally each k-cube has $2^k - k$ switches for connecting k nodes. Therefore, there are $(n-1) \times k^n + k^{n-1} \times 2^{k-1} + (2^k - k) \times k^{n-1} = (n-1)k^n + (2^{k-1} + 2^k - k)k^{n-1}$ links in KANTC(k, n). □

Theorem 5. *The diameter of KANTC(k, n) is $2n + k$.*

Proof. The diameter is defined as the maximum distance of the shortest-path between any two nodes. We call the switch connected to the source compute node source switch and the switch connected to the destination compute node destination switch. Respectively called the source (destination) intermediate switch of the source (destination) compute node, the Source (destination) intermediate switch (SIS/DIS). The longest shortest-path of KANTC consists of three parts: the source switch to the SIS, the SIS to the DIS, and the DIS to the destination switch. When a k-cube has multiple evenly distributed SIS/DIS to choose from, the longest and shortest path is half the diameter of the k-cube, i.e., $k/2$. In KANTC(k, n), the longest shortest-path between two intermediate switches is the path where their Nearest Common Ancestor (NCA) is at stage $n-1$. SIS will send packets at a distance of n-1 to their NCA and $n-1$ from NCA to DIS. Note that the distance between the source (and destination) compute node and the source (and destination) switch is 2. Therefore, the diameter of KANTC(k, n) is $k + 2(n-1) + 2 = 2n + 2k$. □

D. Topological Properties of MiKANTC

Theorem 6. *The radix of the switch of MiKANTC(k, n) is $2k$.*

Proof. MiKANTC(k, n) is divided into two parts, the upper part MiKANT is composed and the lower part consists of $2k^{n-2}$ k-cubes. The radix of switches of MiKANT is $2k$. In the k-cube part, each intermediate switch is connected to k switches of MiKANT and k other k-cube switches, and each switch connecting a computational node is connected to the other k switches and k computational nodes. The radix of k-cube is also $2k$. That is, the radix of MiKANTC(k, n) is $2k$. □

Theorem 7. *There are $2(2^k - k)k^{n-1}$ compute nodes in MiKANTC (k, n).*

Proof. MiKANTC(k, n) has only the lower half of the cube switches used to connect the computational nodes. There are a total of $2k^{n-2}$ k-cubes, and there are $2^k - k$ switches in a k-cube to connect compute nodes; each switch connects k compute nodes. That is, the total number of computational nodes in MiKANTC(k, n) is $2k^{n-2} \times (2^k - k) \times k = 2(2^k - k)k^{n-1}$ □

Theorem 8. *There exist $(2n-4)k^{n-1} + 2^{k+1}k^{n-2}$ switches in MiKANTC(k, n).*

TABLE I
COMPARISON OF NETWORK TOPOLOGICAL PROPERTIES

Parameters	HC(k)	k-ary n-tree	MiKANT(k,n)	KANTC(k,n)	MiKANTC(k,n)
Nodes	2^k	k^n	$2k^n$	$(2^k-k)k^{n-1}$	$2(2^k-k)k^{n-1}$
Switches	2^k	nk^{n-1}	$(2n-2)k^{n-1}$	$(n-1)k^{n-1}+2^kk^{n-2}$	$(2n-4)k^{n-1}+2^{k+1}k^{n-2}$
Links	$k2^{k-1}$	nk^n	$(2n-1)k^n$	$(n-1)k^n$ $+(2^{k-1}+2^k-k)k^{n-1}$	$(2n-3)k^n$ $+(3\times2^k-2k)k^{n-1}$
Radix/Degree	k	$2k$	$2k$	$2k$	$2k$
Diameter	k	$2n$	$2n$	$2n+k$	$2n+k$

Proof. In the top half, there are $2n-4$ layers with k^{n-1} switches each, and in the bottom half, there are $2k^{n-2}$ k-cubes with 2^k switches each. Thus, there are $(2n-4)\times k^{n-1}+2\times 2^k\times 2k^{n-2}=(2n-4)k^{n-1}+2^{k+1}k^{n-2}$ switches in MiKANTC(k,n). □

Theorem 9. *There exist $(2n-3)k^n+(3\times 2^k-2k)k^{n-1}$ links in MiKANTC(k,n).*

Proof. In the upper half of MiKANTC(k,n), there are $2n-3$ layers with k^{n-1} switches per layer, and each switch provides k connections. In the lower half, there are $2k^{n-2}$ k-cubes, each k-cube has $k2^{k-1}$ links, and finally, each k-cube has 2^k-k switches for connecting k. Therefore, MiKANTC(k,n) has $(2n-3)\times k^n+k^{n-1}\times 2^k+2(2^k-k)\times k^{n-1}=(2n-3)k^n+(3\times 2^k-2k)k^{n-1}$ links. □

Theorem 10. *The diameter of MiKANTC(k,n) is $2n+k$.*

Proof. In MiKANTC(k,n), the longest shortest-path is the path between two nodes with $Ci=1$ in the same group with $0\leq i\leq n-1$, their NCAs is located at $n-2$ stage of the other group. The path consists of three parts: 1. source switch to the SIS. 2. SIS to the DIS. 3. DIS to the destination switch. The length of the longest shortest-path for both 1. and 3. is $k/2$, and the length of the longest shortest path for 2. is $2(n-1)$. Therefore, the diameter of MiCAT(k, n) is $k+2(n-1)+2=2n+k$, where $+2$ is the distance between the source (and destination) computational node and the source (and destination) switch. □

Table. I summarizes the topological properties of k-cubes, k-ary n-trees, MiKANT(k,n), KANTC(k,n), and MiKANTC(k,n). As we can see from the table, KANTC and MiKANTC achieve more computational nodes connected with fewer switches and links than the traditional k-ary n-tree and MiKANT by increasing only the diameter of k. In the next section, we will evaluate the detailed performance of the topology.

IV. Routing Algorithm

This section gives the routing algorithms of the KANTC(k,n) and MiKANTC(k,n).

A. Routing Algorithm for KANTC(k,n)

There are many routing algorithms for traditional k-ary n-trees and hypercubes. As mentioned in the previous section, we use a static method to select all intermediate switches and store their IDs in a list $I=[I_{k-1},I_{k-2},...,I_{k-2}]$. So for the lower half of KANTC (the hypercube part), if the source node and the destination node are in different cube's switches, the routing of KANTC(k,n) is to find the shortest path to the intermediate switch. If there are multiple shortest paths, select one at random. If the source node and the destination node are in the same cube's switch, the source node will send the packet directly to the destination node. For the upper part, the basic idea is to find the NCAs of the source intermediate switch and the destination intermediate switch. The route from the source switch to the destination switch can be considered as two parts: SIS to NCA and from NCA to the DIS. There are several paths to the NCA, but the path from the NCA to the destination switch is deterministic.

The routing algorithm of KANTC(k,n) is based on the current switch/node and the destination node ID. We use $W=\langle L_W,W_{n-2},...,W_1,W_0,C_{k-1},...,C_0\rangle$ to represent the current switch/node ID. Packets will be received by destination nodes $T=\langle L_T,T_{n-2},...,T_1,T_0,F_{k-1},...,F_0\rangle$. If $W_{n-2},...,W_{LW}\neq T_{n-2},...,T_{LW}$. This means that the current switch/node has not reached the NCA yet. We need to send the packet to the upper layer switch through the intermediate switch in I. Otherwise, the data packet has reached the NCA, and the routing enters the downward phase. At this stage, routing is deterministic. It must choose to send the packet to the destination switch. In the going to NCA phase, a switch W in the L_W level:

$$W=\langle L_W,W_{n-2},...,W_{LW+1},W_{LW},T_{LW-1},...,T_0,C\rangle$$

send packets to switch U that is closer to T than W

$$U=\langle L_U,W_{n-2},...,W_{LW+1},T_{LW},T_{LW-1},...,T_0,C\rangle$$

where C represents $C_{k-1},...,C_0$, W_{LW} is changed to the T_{LW}. If the L_W of W equals 0, the current switch/node and the destination node are in the same cube. The packet will be sent directly to the destination node T.

This routing algorithm is formally given in **Algorithm 1**, where T^{+}_{LW} is the port label of switch W, which is linked to a switch with a phase equal to $L_W + 1$ (increasing level). Likewise, T^{-}_{LW} is the port label of switch W, which is linked to a switch with a phase equal to $L_W - 1$(decrementing level). The packets will be sent to the NCA of the source and destination switches via T^{+}_{LW}. In the downward phase (reaching the NCA), the packet will be sent to the destination switch via T^{-}_{LW}.

Algorithm 1 **KANTC_Routing**

```
Input: $packet = \langle T, data \rangle$ ;                    /* received packet which will be sent to T*/
$W = \langle L_W, W_{n-2}, ..., W_0, C_{k-1}, ..., C_0 \rangle$ ;              /* current switch/node ID */
$I = [I_{k-1}, I_{k-2}, ..., I_{k-2}]$;                                  /*intermediate cube list*/
$T = \langle L_T, T_{n-2}, ..., T_1, T_0, F_{k-1}, ..., F_0 \rangle$ ;             /* destination node ID */
if ($W_{n-2}, ..., W_{LW} \neq T_{n-2}, ..., T_{LW}$)                             /* going to NCA */
    send packet to IS;                                  /* to the source intermediate switch*/
    if ($C_{k-1}, ..., C_0$ in I)                         /* reached source intermediate switch*/
        send packet to $T^{+}_{LW}$;                                        /* increasing level*/
    else
        if($C_{k-1}, ..., C_0 = 0$)                                         /* going to NCA */
            send packet to $T^{+}_{LW}$;                                    /* increasing level*/
        endif
else                                    /* going to destination intermediate switch from NCA */
    if ($LW > 0$)                                             /* current switch not in level 0 */
        send packet to $T^{-}_{LW}$;                                       /* decreasing level */
    else
        if($W_{n-2}, ..., W_1 = T_{n-2}, ..., T_1$)                /* Destination node in the same cube */
            send packet to T;                                             /* to destination node */
        endif
    endif
endif
```

B. *Routing Algorithm for MiKANTC(k, n)*

MiKANTC(k, n) routing algorithm is based on MiKANT(k, n) and hypercube routing algorithm. Similar to KANTC(k, n), We denote the current switch/node ID by $W = \langle G_W, L_W, W_{n-2}, ..., W_1, W_0, C_{k-1}, ..., C_0 \rangle$. and $T = \langle G_T, L_T, T_{n-2}, ..., T_1, T_0, F_{k-1}, ..., F_0 \rangle$ represents the destination node ID. When W and T are in the same group, the routing algorithm of MiKANTC(k, n) is the same as KANTC(k, n). If W and T are in different groups, the packet will be sent to level $n-2$ of $G_W(T^{+}_{LW})$ through the source switch. Then, from T^{+}_{LW} to the destination switch and finally to the destination node. This routing algorithm is formally given in **Algorithm 2**.

C. *Packet Latency*

We first conducted a small-scale simulation of the routing algorithm and evaluated the average packet latency of KANTC and MiKANTC at $k = 3$ and $n = 4$ by simulation. We evaluate the simulations in a uniform pattern per clock cycle. In uniform traffic, the destination addresses of packets are randomly assigned. For each packet within one clock cycle, if the switch has buffers available, the packet can be sent to another switch/node. Otherwise, the packet will wait for one clock cycle. We set the traffic load λ in the range of 0.05 and 1.00 in units of 0.05. On each clock cycle, $N \times \lambda$ compute nodes send packets to their destination nodes, where N is the number of nodes in the system. The simulation terminates when each destination node receives an average of 200 packets.

Algorithm 2 **MiKANTC_Routing**

```
Input: $packet = \langle T, data \rangle$ ;                    /* received packet which will be sent to T*/
$W = \langle G_W, L_W, W_{n-2}, ..., W_0, C_{k-1}, ..., C_0 \rangle$ ;          /* current switch/node ID */
$I = [I_{k-1}, I_{k-2}, ..., I_{k-2}]$;                                  /*intermediate cube list*/
$T = \langle G_T, L_T, T_{n-2}, ..., T_1, T_0, F_{k-1}, ..., F_0 \rangle$ ;         /* destination node ID */
if($G_W \neq G_T$)                                            /* W, T: different groups */
    send packet to IS;                                  /* to the source intermediate switch*/
    if ($C_{k-1}, ..., C_0$ in I)                         /* reached source intermediate switch*/
        send packet to $T^{+}_{LW}$;                                        /* increasing level*/
    else
        if($C_{k-1}, ..., C_0 = 0$)                                         /* going to NCA */
            send packet to $T^{+}_{LW}$;                                    /* increasing level*/
        endif
else
    if ($W_{n-2}, ..., W_{LW} \neq T_{n-2}, ..., T_{LW}$)                         /* going to NCA */
        send packet to IS;                              /* to the source intermediate switch*/
        if ($C_{k-1}, ..., C_0$ in I)                     /* reached source intermediate switch*/
            send packet to $T^{+}_{LW}$;                                    /* increasing level*/
        else
            if($C_{k-1}, ..., C_0 = 0$)                                    /* reaching NCA */
                send packet to $T^{+}_{LW}$;                                /* increasing level*/
            endif
    else                                /* going to destination intermediate switch from NCA */
        if ($LW > 0$)                                         /* current switch not in level 0 */
            send packet to $T^{-}_{LW}$;                                   /* decreasing level */
        else
            if($W_{n-2}, ..., W_1 = T_{n-2}, ..., T_1$)            /* Destination node in the same cube */
                send packet to T;                                         /* to destination node */
            endif
        endif
    endif
endif
```

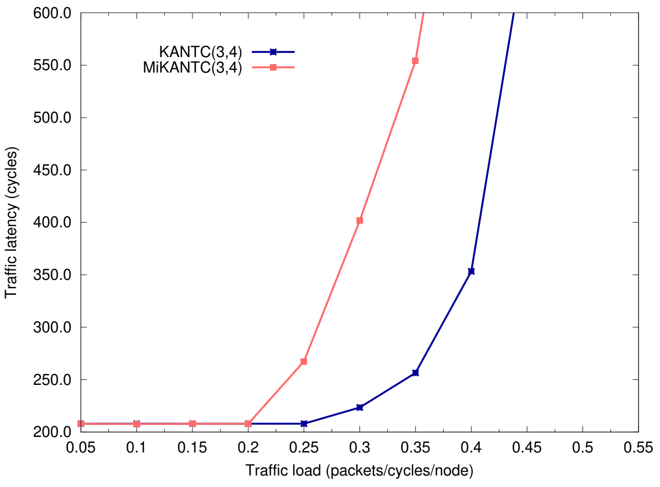


Fig. 5. packet latencies of KANTC$(3, 4)$ and MiKANTC$(3, 4)$

Fig. 5 illustrates the packet latency for KANTC$(3, 4)$ and MiKANTC$(3, 4)$, respectively. The vertical axis represents the average packet delay in clock cycles, and the horizontal axis represents the traffic load λ. The buffer size is 8. We can see that our proposed algorithm works efficiently when the load is below 40%.

V. Performance Evaluation

In this section, we evaluate the cost, performance, and path diversity of KANTC(k, n) and MiKANTC(k, n).

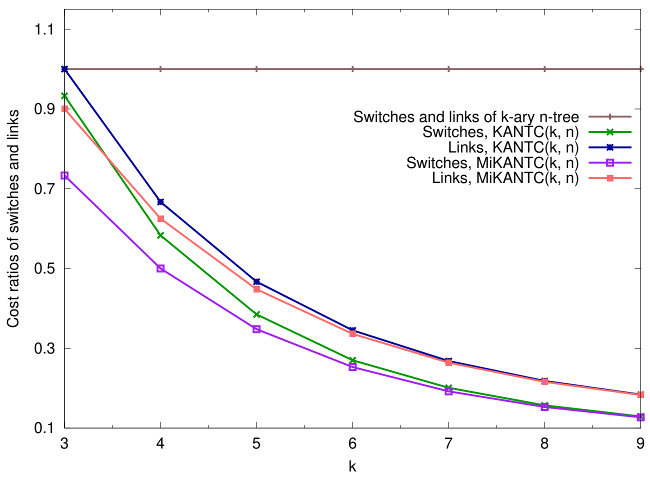


Fig. 6. Cost ratios of switches and links to k-ary n-tree

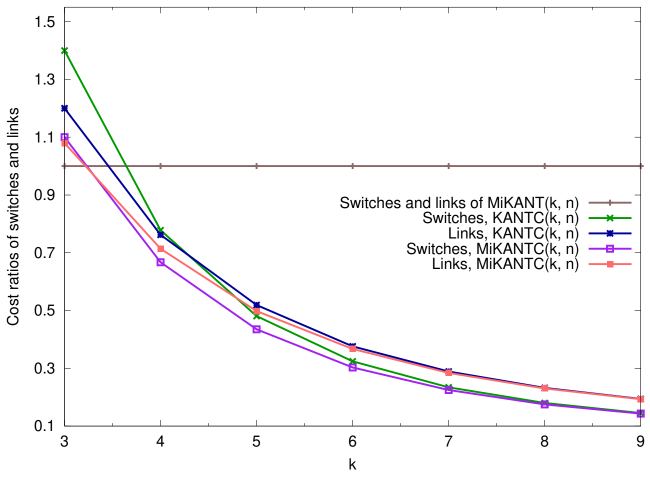


Fig. 7. Cost ratios of switches and links to MiKANT(k, n)

A. Cost Ratio

A k-ary n-tree has k^n nodes and nk^{n-1} switches. That is, a node requires nk^{n-1}/k^n switches. A KANTC(k, n) has $(2^k - k)k^{n-1}$ nodes and $(n-1)k^{n-1} + 2^k k^{n-2}$ switches. The switch cost ratio of a KANTC(k, n) to a k-ary n-tree is

$$\frac{((n-1)k^{n-1} + 2^k k^{n-2})/(2^k - k)k^{n-1}}{nk^{n-1}/k^n} = \frac{2^k - k + kn}{n(2^k - k)}$$

The link cost ratio of a KANTC(k, n) to k-ary n-tree is

$$\frac{((n-1)k^n + (2^{k-1} + 2^k - k)k^{n-1})/(2^k - k)k^{n-1}}{nk^{n-1}/k^n}$$

$$= \frac{(n-2)k^2 + 3 \times 2^{k-1}k}{n(2^k - k)}$$

Similarly, we can emanate expressions for computing the switch and link cost ratio of MiKANT(k, n). We plot the switch and link cost ratios of KANTC(k, n) and MiKANTC(k, n) versus k-ary n-tree and MiKANT(k, n) in fig. 6 and fig. 7, respectively. For convenience, we set $n = k$ in both figures. We can see that both KANTC(k, n) and MiKANTC(k, n) have lower switch and link cost ratios than k-ary n-tree and MiKANT(k, n) when $n = k \geq 4$. For $k = n = 8$, KANTC saves 84.27% switches and 78.23% links than k-ary n-tree; 82.03% switches and 76.77% links than MiKANT(k, n). Compared with k-ary n-tree, MiKANTC saves 84.68% of switches and 78.43% of links; compared with MiKANT(k, n), it saves 82.49% of switches and 76.99% of links.

B. Relative Cost Performance

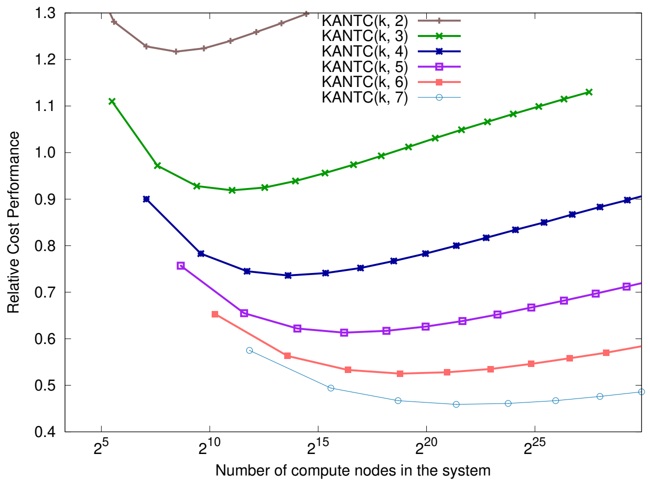


Fig. 8. RCP comparison of KANTC(k, n)

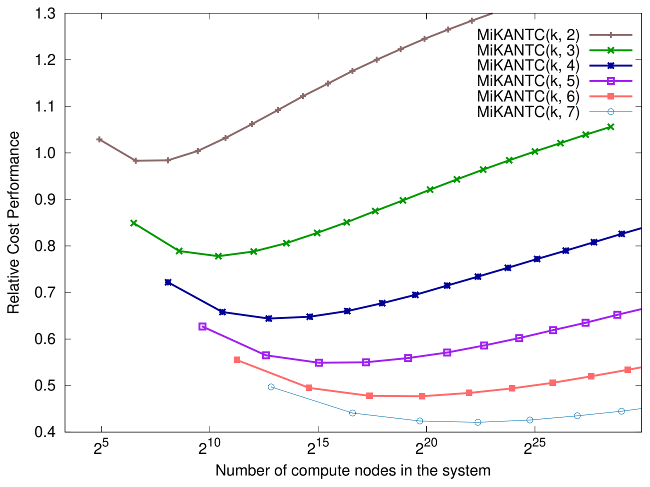


Fig. 9. RCP comparison of MiKANTC(k, n)

Suppose we build the network topology KANTC(k, n) and a MiKANTC(k, n) with two given node numbers $N1 = (2^k - k)k^{n-1}$ and $N2 = 2(2^k - k)k^{n-1}$, respectively. We would like to know how to decide the values of k and n

so that KANTC(k,n) and MiKANTC(k,n) can achieve high performance at a low cost. Generally, a larger k will increase the switching cost, and a larger n will increase communication time.

To make a good trade-off between the cost and performance of KANTC(k,n) and MiKANTC(k,n), we define the $relativecostperformance(RCP)$ of the hypercube as follows:

$$RCP_1 = 2k \times (2n+k)(\log_2 N_1/p + p) \times (\log_2 N_1/p + 2)$$

$$RCP_2 = 2k \times (2n+k)(\log_2 N_2/p + p) \times (\log_2 N_2/p + 2)$$

Among them, RCP_1 and RCP_2 refer to the RCP of KANTC(k,n) and the RCP of MiKANTC(k,n), respectively; $2k$ is the radix of KANTC(k,n) and MiKANTC(k,n), which affects the cost of hardware and software; $2n+2k$ is the diameter of KANTC(k,n) and MiKANTC(k,n), which affects the communication performance; $\log_2 N/p$ is the size of the hypercube; p represents the number of ports in the router connecting computing nodes. The radix and the diameter of a k-cube are k, but in a practical implementation, a router has p ports for connecting compute nodes. The diameters of KANTC(k,n) and MiKANTC(k,n) contain links connecting compute nodes and switches. For a fair comparison, we let the diameter of the k-cube be $k+2$.

Fig. 8 and fig. 9 illustrate the RCPs of KANTC(k,n) and MiKANTC(k,n) for k-cubes in the case of $2 \leq n \leq 7$, $p = 1$, respectively. For a given n, we vary the value of k to implement systems of different scales. Lower values in the curve mean the system achieves higher performance at lower hardware cost. A value less than 1 means the system is better than a hypercube. For example, when we build a system of 5412096 nodes with MiKANTC(6, 7), the RCP is 0.421. For a system of a given size, we can choose appropriate k and n numerically so that the RCP of the system will be lower.

C. Path Diversities

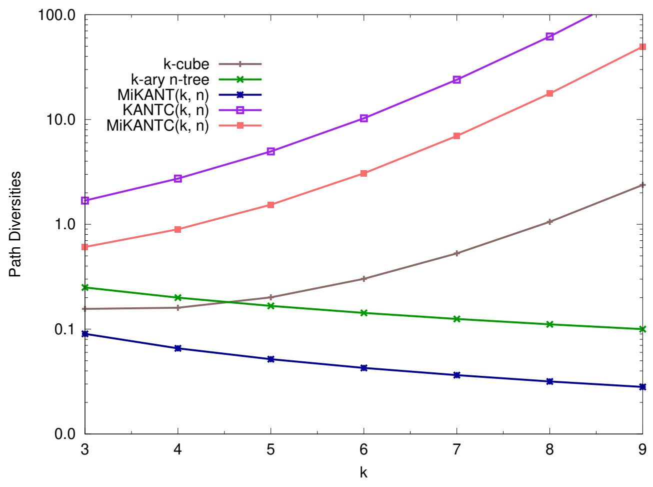


Fig. 10. Path diversities

In a network topology, path diversity is an important property. A high path diversity topology will provide lots of paths between any two nodes, providing more fault tolerance choices. We define the path diversity (PD) of the network as follows.

$$PD = \frac{\widetilde{P}}{N}$$

where $\widetilde{P}$ is the average number of shortest paths and N is the number of nodes in the system.

First, we need to calculate the PD of the k-cube. For any two nodes with distance i in the k-cube ($0 \leq i \leq k$), there are $i!$ shortest paths. For any node in the k-cube, there is a C_k^1 node at a distance of 1, and there is 1! shortest path, there are C_k^2 nodes at distance 2, there are 2! shortest paths, there are C_k^i nodes at distance i, there is $i!$ shortest paths, there are C_k^k nodes at distance k, there are $k!$ shortest paths. $\widetilde{P}$ of k-cube is

$$\tilde{P}_{cube} = (\sum_{i=1}^{k} C_k^i \times i!)/2^k = \sum_{i=1}^{k} \frac{k!}{i! \times 2^k}$$

The path diversity PD of a k-cube is

$$PD_{cube} = \frac{\tilde{P}_{cube}}{N_{cube}} = \sum_{i=1}^{k} \frac{k!}{i! \times 4^k}$$

In k-ary n-tree, path diversity is only related to switches. The path is fixed from a compute node (switch) to a switch (compute node). The number of shortest paths depends on the level of NCA. If the NCA is at level i, there are k^i paths for the source switch to send packets to one of the NCAs. The average number of paths can be calculated as follows. For $n \geq 2$, each switch has $k-1$ switches with k^1 paths, k^2-k switches with k^2 paths, ..., $k^{n-1}-k^{n-2}$ switches with k^{n-1} paths. That is, the $\widetilde{P}$ of the k-ary n-tree is

$$\tilde{P}_{kant} = [\sum_{i=1}^{n-1} k^i(k^i - k^{i-1})]/k^{n-1} = \frac{k^n - 1/k^{n-2}}{k+1}$$

The path diversity PD of the k-ary n-tree is

$$PD_{kant} = \frac{\tilde{P}_{kant}}{N_{kant}} = \frac{1 - 1/k^{2n-2}}{k+1}$$

The path diversity of MiKANT(k,n) is similar to a k-ary n-tree. There are $k-2$ paths for the source switch to send packets to different groups of $n-2$ switches. Therefore, $\widetilde{P}$ of MiKANT(k,n) is

$$\tilde{P}_{mikant} = [\sum_{i=1}^{n-1} k^i(k^i - k^{i-1}) + k^{n-2} \times k^{n-1}]/2k^{n-1}$$

$$= \frac{k^n - 1/k^{n-2}}{2(k+1)} + \frac{k^{n-2}}{2}$$

The path diversity PD of MiKANT(k,n) is

$$PD_{mikant} = \frac{\tilde{P}_{mikant}}{N_{mikant}} = \frac{1 - 1/k^{2n-2}}{4(k+1)} + \frac{1}{4k^2}$$

To compute $\widetilde{P}$ for KANTC(k, n), we have to consider the following two cases. Whether the source node and destination node are in the same k-cube, The same probability is $1/k^{n-2}$, since there are k intermediate nodes to choose from in the hypercube part of KANTC, $\tilde{P}_0 = k\tilde{P}_{cube}$. The probability of not being the same is $1 - 1/k^{n-2}$, $\tilde{P}_1 = k^2\tilde{P^2}_{cube} \times \tilde{P}_{kant}$. So,$\widetilde{P}$ of KANTC(k, n) is

$$\tilde{P}_{kantc} = \tilde{P}_1(1 - \frac{1}{k^{n-2}}) + \tilde{P}_0\frac{1}{k^{n-2}}$$

$$= k^2\tilde{P^2}_{cube}\tilde{P}_{kant} + \frac{k\tilde{P}_{cube} - \tilde{P^2}_{cube}\tilde{P}_{kant}}{k^{n-4}}$$

The path diversity PD of KANTC(k, n) is

$$PD_{kantc} = \frac{\tilde{P}_{kantc}}{N_{kantc}}$$

$$= \frac{k^2\tilde{P^2}_{cube}\tilde{P}_{kant}}{N_{kantc}} + \frac{k\tilde{P}_{cube} - \tilde{P^2}_{cube}\tilde{P}_{kant}}{N_{kantc}k^{n-4}}$$

$\widetilde{P}$ of MiKANTC(k, n) is similar to KANTC(k, n). It replaces $\tilde{P}_{kant}$ with $\tilde{P}_{mikant}$, and changes the probability of the same to $1/(2k^{n-2})$ and the probability of the same to $(2k^{n-2} - 1)/(2k^{n-2})$. $\widetilde{P}$ of MiKANTC(k, n) is

$$\tilde{P}_{mikantc} = k^2\tilde{P^2}_{cube}\tilde{P}_{mikant} + \frac{k\tilde{P}_{cube} - \tilde{P^2}_{cube}\tilde{P}_{mikant}}{2k^{n-4}}$$

The path diversity PD of MiKANTC(k, n) is

$$PD_{mikantc} = \frac{\tilde{P}_{mikantc}}{N_{mikantc}}$$

$$= \frac{k^2\tilde{P^2}_{cube}\tilde{P}_{mikant}}{N_{mikantc}} + \frac{k\tilde{P}_{cube} - \tilde{P^2}_{cube}\tilde{P}_{mikant}}{2N_{mikantc}k^{n-4}}$$

Fig. 10 plots the path diversities of the k-cube, k-ary n-tree, MiKANT(k, n), KANTC(k, n), and MiKANTC(k, n) with $n = 8$. We can see that the link diversity of KANTC(k, n) and MiKANTC(k, n) is better compared to other topologies due to the increase in paths, which means that they have better routing performance and fault tolerance.

VI. Conclusions

This paper proposes two new interconnection networks, KANTC and MiKANTC. Both proposed networks achieve high performance and low cost. We propose routing algorithms and test it under small-scale simulations, then evaluate the two networks' cost ratios, relative cost performance, and path diversities. The results show that KANTC and MiKANTC have lower hardware costs and higher path diversity than the traditional fat-tree. So our future work is to propose better performing routing algorithms and simulate them on larger scale networks.

References

[1] W. Zheng, "Research trend of large-scale supercomputers and applications from the top500 and gordon bell prize," *Science China Information Sciences*, vol. 63, no. 7, pp. 1–14, 2020.

[2] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[3] J. Wells, B. Bland, J. Nichols, J. Hack, F. Foertter, G. Hagen, T. Maier, M. Ashfaq, B. Messer, and S. Parete-Koon, "Announcing supercomputer summit," Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), Tech. Rep., 2016.

[4] C. B. Stunkel, R. L. Graham, G. Shainer, M. Kagan, S. Sharkawi, B. Rosenburg, and G. Chochia, "The high-speed networks of the summit and sierra supercomputers," *IBM Journal of Research and Development*, vol. 64, no. 3/4, pp. 3–1, 2020.

[5] F. Petrini and M. Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Proceedings 11th international parallel processing symposium*. IEEE, 1997, pp. 87–93.

[6] C. G. Requena, F. G. Villamón, M. E. G. Requena, P. J. L. Rodríguez, and J. D. Marín, "Ruft: Simplifying the fat-tree topology," in *2008 14th IEEE International Conference on Parallel and Distributed Systems*. IEEE, 2008, pp. 153–160.

[7] D. F. B. Garzón, C. G. Requena, M. E. Gómez, P. López, and J. Duato, "A family of fault-tolerant efficient indirect topologies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 927–940, 2015.

[8] D. Ludovici, F. Gilabert, C. Requena, M. Gmez, P. Lpez, G. Gaydadjiev, and J. Duato, "Butterfly vs. unidirectional fat-trees for networks-on-chip: Not a mere permutation of outputs," in *Proc. 3rd Workshop Interconnection Netw. Archit. On-Chip Multi-Chip*. Citeseer, 2009.

[9] Z. Wang, J. Xu, X. Wu, Y. Ye, W. Zhang, M. Nikdast, X. Wang, and Z. Wang, "Floorplan optimization of fat-tree-based networks-on-chip for chip multiprocessors," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1446–1459, 2012.

[10] Y. Li and W. Chu, "Mikant: A mirrored k-ary n-tree for reducing hardware cost and packet latency of fat-tree and clos networks," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2018, pp. 1643–1650.

[11] Y. Wang and Y. Li, "Link fault tolerant routing algorithms in mirrored k-ary n-tree interconnection networks," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2020, pp. 237–241.

[12] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Transactions on computers*, vol. 37, no. 7, pp. 867–872, 1988.

[13] D. Arai and Y. Li, "Generalized-star cube: A new class of interconnection topology for massively parallel systems," in *2015 Third International Symposium on Computing and Networking (CANDAR)*. IEEE, 2015, pp. 68–74.

[14] Y. Wang and Y. Li, "Hybrid interconnection networks for reducing hardware cost and improving path diversity based on fat-trees and hypercubes," in *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2021, pp. 254–260.