

# MiKANT: A Mirrored K-Ary N-Tree for Reducing Hardware Cost and Packet Latency of Fat-Tree and Clos Networks

Yamin Li

*Department of Computer Science  
Hosei University  
Tokyo 184-8584 Japan  
yamin@hosei.ac.jp*

Wanming Chu

*Division of Information Systems  
University of Aizu  
Aizu-Wakamatsu 965-8580 Japan  
w-chu@u-aizu.ac.jp*

**Abstract**—Interconnection networks based on the fat-tree topology are widely used in high-performance parallel super-computers. In a classical fat-tree, the radix of root switches is less than that of other switches. Fat-tree is a folded version of a Clos network. A Clos network uses the same radix switches in all stages. However, fat-tree or Clos network has a high switch cost and great packet latency. This paper proposes a variant of the fat-tree, named Mirrored  $k$ -ary  $n$ -tree (MiKANT), that doubles the number of compute nodes of the fat-tree by adding a few switches and making all the switches have a same radix. Compared to the classical fat-tree and Clos network, MiKANT not only reduces the numbers of switches and links so that it can be implemented at lower hardware cost, but also makes the network average distance shorter for achieving higher communication performance. We describe the structure of MiKANT, examine its topological properties, give a minimal per-hop deterministic routing algorithm, and evaluate the cost performance. Our results show that MiKANT reduces the average distance by about 0.5, saves 6.3% to 25.0% links and 12.5% to 50.0% switches, and improves performance of 9.1% to 41.4%, compared to the classical fat-tree, when  $n$  is in the range of 2 and 8. Our simulation results also show that MiKANT achieves much lower average packet latencies than the Clos network.

**Index Terms**—interconnection network, fat-tree,  $k$ -ary  $n$ -tree, switch, diameter, radix, routing algorithm, packet latency, cost performance evaluation

## I. INTRODUCTION

The fat-tree [1] is one of the most commonly used topologies of the interconnection networks in commercial super-computers. Unlike the traditional tree structure, fat-tree has a structure similar to the actual tree. In other words, fat-tree gets fatter (thicker) near the root. Compute nodes are attached only to the switches of the leaf stage (other stages contain only switches). In the fat-tree with a one root, the number of ports of the switches increases as the stage goes toward the root. To keep a fixed switch radix, some fat-tree alternatives which have multiple roots were proposed. Furthermore, to parameterize the multiple-root fat-trees, Petrini and Vanneschi proposed a  $k$ -ary  $n$ -tree [2], where  $k$  is the arity or the number of links of a switch that connects to the previous or next stage; i.e., the switch radix is  $2k$ , and  $n$  is the number of stages. Qian et al. presented routing algorithms in fat-tree data center network [3].

The fat-tree is a folded version of a Clos network [4] which was originally designed for non-blocking telecommunication

circuit switching. Clos network is a multistage interconnection network (MIN) that uses small-scale crossbar as the building blocks. MIN showed that it outperforms a single large-scale crossbar when the number of nodes becomes large, because in the crossbar, the number of crosspoints increases quadratically with the number of ports. Google has adopted the Clos network in its data center network architecture design [5].

Fat-tree and Clos network provide high path diversity but meanwhile require a higher number of switches with a non-negligible wiring complexity and have a greater packet latency than other low cost MINs. The increased switch cost and packet latency both stem from the need to route packets first to an arbitrary middle stage or root switch and then to their ultimate destination [6].

In order to reduce the hardware cost of the switches, Gómez et al. proposed a reduced unidirectional fat-tree (RUFT) [7], [8]. Because the links in RUFT are unidirectional, all packets must traverse from the first stage switches to the last stage switches and then traverse back to the compute nodes along with long links. Ludovici et al. showed that RUFT is a more powerful alternative than the traditional Butterfly for the implementation of the network-on-chip (NoC) [9]. Wang et al. pointed out that the floorplan design of the fat-tree-based NoC is very challenging because of the complexity of topology, and proposed a method to optimize the fat-tree floorplan which can effectively reduce the number of crossings and minimize the interconnect length [10]. Navaridas et al. proposed a thin tree [11] that removes some switches and links from the  $k$ -ary  $n$ -tree, for reducing the hardware cost of the switches.

The main contribution of this paper is to propose a Mirrored  $k$ -ary  $n$ -tree (MiKANT) to reduce the complexity of the fat-tree by connecting more compute nodes with less switches and links, and as a result, to reduce the hardware cost and to make it easier to be implemented than the fat-tree and bidirectional Clos network. Meanwhile, MiKANT shortens the average distance to reduce the communication time for achieving high performance. We also give a minimal per-hop deterministic routing algorithm and a method to determine the values of  $k$  and  $n$  for a given system size. Our analytical and synthetic simulation results show that MiKANT outperforms the classical  $k$ -ary  $n$ -tree and Clos network, in terms of hardware cost, average distance, and average packet latency.

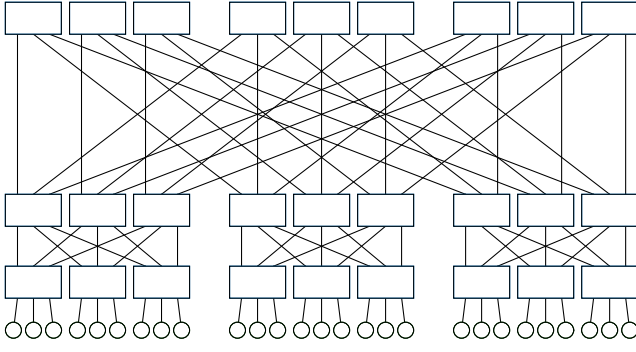


Fig. 1. A classical 3-ary 3-tree

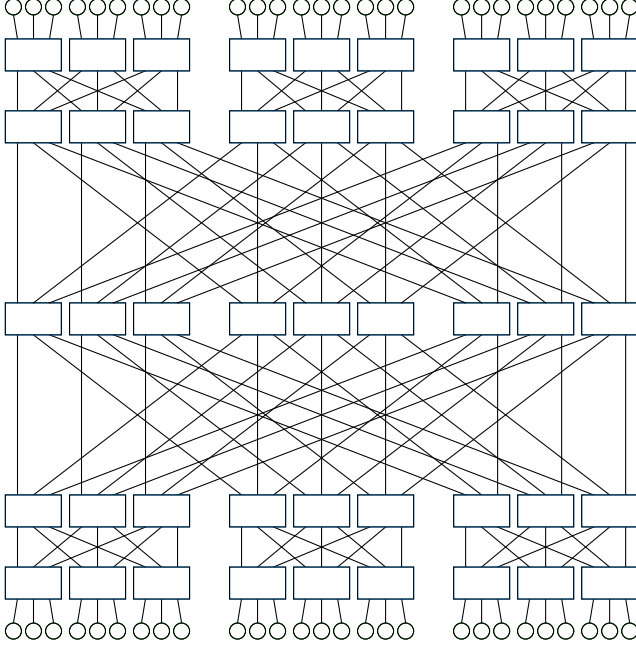


Fig. 2. A bidirectional Clos 3-ary 3-tree

The rest of the paper is organized as follows. Section II describes the structure of MiKANT. Section III examines its topological properties. Section IV proposes a minimal per-hop deterministic routing algorithm. Section V evaluates the cost performance by the analytical and synthetic simulations. And Section VI concludes the paper.

## II. MIRRORED K-ARY N-TREE

MiKANT is motivated from the classical  $k$ -ary  $n$ -tree and Clos  $k$ -ary  $n$ -tree. The classical  $k$ -ary  $n$ -tree is a special case of fat-trees. The number of stages of the classical  $k$ -ary  $n$ -tree is  $n$  and in each stage, there are  $k^{n-1}$  switches. This also means that there are  $k^{n-1}$  roots.  $k$  compute nodes are attached to each switch in the leaf stage. The total number of nodes is  $k^{n-1} \times k = k^n$ . The diameter is  $2n$ , including the distance between switch and compute node. Fig. 1 shows a classical 3-ary 3-tree where the rectangles represent switches and the circles represent compute nodes. There are  $k^n = 27$

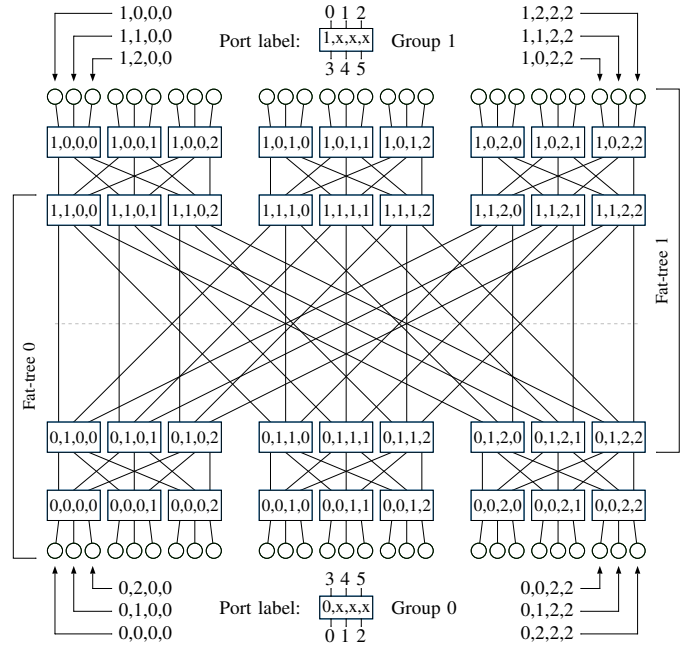


Fig. 3. A Mirrored 3-ary 3-tree

nodes and  $nk^{n-1} = 27$  switches. In the classical  $k$ -ary  $n$ -tree, switches in the root stage have less ports than other switches. For example, the switches in the top row of Fig. 1 are radix-3 and the others are radix-6.

A Clos network [4] consists of an odd number of stages. Particularly, a Clos  $k$ -ary  $n$ -tree has  $2n - 1$  stages and each stage has  $k^{n-1}$  switches. The original Clos network uses *unidirectional* channels. An injection port and an ejection port are connected to a same physical terminal node. Fig. 2 illustrates a bidirectional Clos 3-ary 3-tree that uses *bidirectional* channels, and the nodes on the top and bottom are distinct.

A Mirrored  $k$ -ary  $n$ -tree, denoted as MiKANT( $k, n$ ), has  $2n - 2$  stages; there are  $k^{n-1}$  switches in each stage; and each switch has  $2k$  bidirectional ports.

Each switch is labeled as  $\langle G, L, D \rangle$ , where  $G$  indicates the *group* with  $G \in \{0, 1\}$ ,  $L$  (level) indicates the *stage* with  $L \in \{0, \dots, n-2\}$ , and  $D = D_{n-2}, D_{n-3}, \dots, D_1, D_0$  is an  $(n-1)$ -tuple  $\{0, 1, \dots, k-1\}^{n-1}$  which identifies the switches inside stage  $L$  of group  $G$ . A switch

$$\langle G, L, D_{n-2}, \dots, D_{L+1}, D_L, D_{L-1}, \dots, D_0 \rangle$$

will connect to switches

$$\langle G, L+1, D_{n-2}, \dots, D_{L+1}, *, D_{L-1}, \dots, D_0 \rangle$$

if  $0 \leq L \leq n-3$ ; otherwise ( $L = n-2$ ) to switches

$$\langle \bar{G}, L, *, D_{n-3}, \dots, D_1, D_0 \rangle$$

where  $* \in \{0, 1, \dots, k-1\}$  and  $\bar{G}$  is the bit-inversion of  $G$ . Because we define the stage  $L$  as  $0 \leq L \leq n-2$ , we discuss MiKANT( $k, n$ ) with a restriction of  $n \geq 2$ . For  $n = 1$ , there is only one switch and  $2k$  compute nodes are connected to the switch. For example, a stage 2 switch of group 0  $w =$

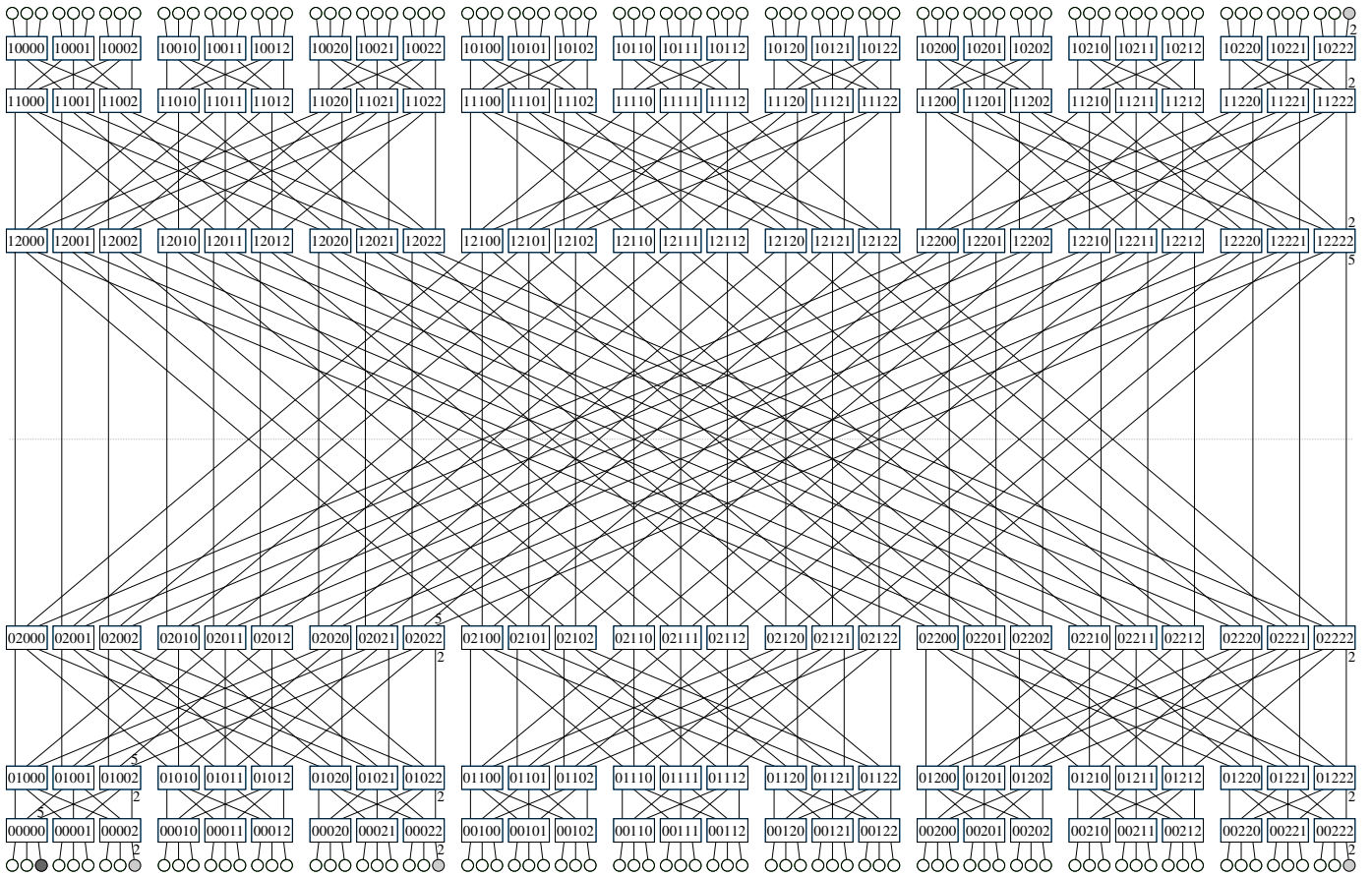


Fig. 4. A Mirrored 3-ary 4-tree

$\langle 0, 2, 0, 0, 0 \rangle$  in a  $\text{MiKANT}(3, 4)$  ( $k = 3$  and  $n = 4$ ) connects to switches  $\langle 1, 2, 0, 0, 0 \rangle$ ,  $\langle 1, 2, 1, 0, 0 \rangle$ , and  $\langle 1, 2, 2, 0, 0 \rangle$  for  $L = n - 2 = 2$ ; and the following three switches connect to  $w$ :  $\langle 0, 1, 0, 0, 0 \rangle$ ,  $\langle 0, 1, 0, 1, 0 \rangle$ , and  $\langle 0, 1, 0, 2, 0 \rangle$  for  $L = 1$ .

Each compute node is defined as  $\langle G, C \rangle$ , where  $G$  is the group with  $G \in \{0, 1\}$ , and  $C = C_{n-1}, C_{n-2}, \dots, C_1, C_0$  is an  $n$ -tuple  $\{0, 1, \dots, k-1\}^n$ . There is a link between a switch

$$\langle G, 0, D_{n-2}, \dots, D_1, D_0 \rangle$$

and a compute node

$$\langle G, C_{n-1}, C_{n-2}, \dots, C_1, C_0 \rangle$$

if  $D_i = C_i$  for all  $i \in \{n-2, \dots, 1, 0\}$ .

Fig. 3 shows a  $\text{MiKANT}(3, 3)$ . There are  $2(n-1) = 4$  stages and each stage has  $k^{n-1} = 9$  switches. The upper two stages are of group 1 and lower two stages are of group 0. The number of compute nodes is  $2k^n = 54$ . Compared to the bidirectional Clos 3-ary 3-tree, shown in Fig. 2, which also has 54 nodes,  $\text{MiKANT}(3, 3)$  removed the center stage switches. The number of switches is decreased from 45 to 36. Compared to the classical 3-ary 3-tree, shown in Fig. 1, which has 27 nodes and 27 switches,  $\text{MiKANT}(3, 3)$  doubles the number of nodes but only adds 9 switches. We can also see that the group 0 and the stage 1 switches of group 1 form a classical 3-ary

3-tree; and the group 1 and the stage 1 switches of group 0 form another classical 3-ary 3-tree.

Fig. 4 shows a  $\text{MiKANT}(3, 4)$ . There are  $2(n-1) = 6$  stages and each stage has  $k^{n-1} = 27$  switches. The number of compute nodes is  $2k^n = 162$ . In contrast, the Clos 3-ary 4-tree has 7 stages.

Generally,  $\text{MiKANT}$  consists of an even number of stages. The switches in stage  $n-2$  of group 1 can be considered as the root switches of group 0; similarly, the switches in stage  $n-2$  of group 0 can be considered as the root switches of group 1. That is, the switches in the center two stages are shared by the two classical  $k$ -ary  $n$ -trees.

Although  $\text{MiKANT}(k, n)$  reduced switches and links, when the source node and destination node are of a same group (group 0 or 1),  $\text{MiKANT}(k, n)$  acts as the same as  $k$ -ary  $n$ -tree (fat-tree). This keeps the non-blocking property of the fat-tree. On the other hand, when the source node and destination node are of distinct groups,  $\text{MiKANT}(k, n)$  reduces both the path length and path diversity. Reducing path diversity will lose the non-blocking property but there are still multiple paths between the source and destination nodes. For some applications, there will be few performance drawbacks. Note that even in this case, we can also route a packet first to an arbitrary root switch of the destination node and then to the ultimate destination node.

### III. TOPOLOGICAL PROPERTIES OF MiKANT

This section gives topological properties of MiKANT( $k, n$ ).

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

*Proof:* (1) The switch  $\langle G, 0, D_{n-2}, \dots, D_1, D_0 \rangle$  in stage 0 has  $k$  links connecting to compute nodes and  $k$  links connecting to switches  $\langle G, 1, D_{n-2}, \dots, D_1, i \rangle$  in stage 1 with  $0 \leq i \leq k-1$ . (2) The switch  $\langle G, n-2, D_{n-2}, D_{n-3}, D_{n-4}, \dots, D_0 \rangle$  in stage  $n-2$  has  $k$  links connecting to switches  $\langle \bar{G}, n-2, i, D_{n-3}, D_{n-4}, \dots, D_0 \rangle$  with  $0 \leq i \leq k-1$  and there are  $k$  links connecting to switches  $\langle G, n-3, D_{n-2}, i, D_{n-4}, \dots, D_0 \rangle$  in stage  $n-3$  with  $0 \leq i \leq k-1$ . (3) The switch  $\langle G, L, D_{n-2}, \dots, D_{L+1}, D_L, D_{L-1}, D_{L-2}, \dots, D_0 \rangle$  in stage  $L$  ( $L \neq 0$  and  $L \neq n-2$ ) has  $k$  links connecting to switches  $\langle G, L+1, D_{n-2}, \dots, D_{L+1}, i, D_{L-1}, D_{L-2}, \dots, D_0 \rangle$  with  $0 \leq i \leq k-1$  and there are  $k$  links connecting to switches  $\langle G, L-1, D_{n-2}, \dots, D_{L+1}, D_L, i, D_{L-2}, \dots, D_0 \rangle$  in stage  $L-1$  with  $0 \leq i \leq k-1$ . Therefore the radix of MiKANT( $k, n$ ) is  $k+k=2k$ . ■

*Theorem 2:* The diameter of MiKANT( $k, n$ ) is  $2n$ .

*Proof:* The diameter is defined as the maximum distance of the shortest-path between any two nodes. We count the distance from a node to its connected switch as a 1. In MiKANT( $k, n$ ), the longest shortest-path is the path between two nodes of a same group and their nearest common ancestor (NCA) is located in the stage  $n-2$  of the other group. The one-way distance between a node to the NCA is  $1+(n-2)+1$ . The first 1 is the distance between the node and its connected switch in stage 0;  $n-2$  is the distance between the switch in stage 0 and the switch in stage  $n-2$  of the same group; and the last 1 is the distance between the stage  $n-2$  switch and the NCA. Therefore, the diameter is  $2(1+(n-2)+1)=2n$ . ■

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

*Proof:* The switch ID in stage  $L$  of group  $G$  is  $\langle G, L, D_{n-2}, \dots, D_{L+1}, D_L, D_{L-1}, \dots, D_0 \rangle$ . The value of each  $D_i$  can be  $0, 1, \dots, k-1$  for  $i=0, 1, \dots, n-2$ . This means that the number of switches in one stage of group  $G$  is  $k^{n-1}$ . There are two groups ( $0 \leq G \leq 1$ ) and each group has  $(n-1)$  stages ( $0 \leq L \leq n-2$ ). Therefore, there are  $2(n-1)k^{n-1}$  switches in MiKANT( $k, n$ ). ■

*Theorem 4:* There are  $2k^n$  nodes in MiKANT( $k, n$ ).

*Proof:* Because a stage 0 switch  $\langle G, 0, D_{n-2}, \dots, D_1, D_0 \rangle$  of each group has  $k$  nodes connected, and there are  $k^{n-1}$  such switches in each group, the total number of nodes is  $2 \times k^{n-1} \times k = 2k^n$ . ■

*Theorem 5:* There are  $(2n-1)k^n$  links in MiKANT( $k, n$ ).

*Proof:* The number of links here contains also the links connecting nodes and switches. Within each group, there are  $n-1$  stages and  $k^{n-1}$  switches in each stage. Each switch contributes  $k$  links. Therefore, there are  $(n-1) \times k^{n-1} \times k = (n-1)k^n$  links inside a group. Also, there are  $k^{n-1} \times k = k^n$  links that connect switches of distinct groups. Therefore, the total number of links in MiKANT( $k, n$ ) is  $2(n-1)k^n + k^n = (2n-1)k^n$ . ■

*Theorem 6:* The average distance  $\tilde{D}_m$  of MiKANT( $k, n$ ) is  $2n-1/(k-1)+1/((k-1)k^n)-1/2$ .

*Proof:* The average distance of MiKANT( $k, n$ ) can be calculated as below. For  $n \geq 2$ , each node has  $k-1$  nodes at distance 2,  $k^2-k$  nodes at distance 4,  $\dots$ ,  $k^n-k^{n-1}$  nodes at distance  $2n$ , of the same group, and  $k^n$  nodes at distance  $2n-1$  of different groups. That is, the average distance

$$\begin{aligned} \tilde{D}_m &= \left[ \sum_{i=1}^n 2i(k^i - k^{i-1}) + (2n-1)k^n \right] / (2k^n) \\ &= 2n - \frac{1}{k-1} + \frac{1}{(k-1)k^n} - \frac{1}{2} \end{aligned} \quad (1)$$

In contrast, the average distance of the bidirectional Clos  $k$ -ary  $n$ -tree is

$$\tilde{D}_s = 2n - \frac{1}{k-1} + \frac{1}{(k-1)k^n} \quad (2)$$

And the average distance of the classical  $k$ -ary  $n$ -tree is

$$\tilde{D}_c = 2n - \frac{2}{k-1} + \frac{2}{(k-1)k^n} \quad (3)$$

*Theorem 7:* The bisection width of MiKANT( $k, n$ ) is  $k^n/2$ .

*Proof:* The bisection width is defined as the smallest number of links we have to cut in order to separate the network into two parts of the same number of nodes. Here we only take the case of even  $k$  to calculate the bisection width. We divide MiKANT( $k, n$ ) into two parts A and B in such a way that part A contains switches with IDs of  $\langle G, L, A_{n-2}, A_{n-3}, \dots, A_1, A_0 \rangle$  for  $0 \leq A_i \leq k/2-1$  and part B contains switches with IDs of  $\langle G, L, B_{n-2}, B_{n-3}, \dots, B_1, B_0 \rangle$  for  $k/2 \leq B_i \leq k-1$ . Then only the links in between stage  $n-2$  of group 0 and stage  $n-2$  of group 1 may connect switches of part A and switches of part B. The number of such links is  $k/2 \times k^{n-1}$  where the last term is the number of switches in a stage, and the first term,  $k/2$ , indicates that for one switch, there are half links connecting to switches of the different part. Therefore, the bisection width of MiKANT( $k, n$ ) is  $k/2 \times k^{n-1} = k^n/2$ . ■

Note that the bisection width of MiKANT( $k, n$ ) is less than the number of links in between group 0 and group 1 which is  $k \times k^{n-1} = k^n$ . Table I summarizes the topological properties of classical  $k$ -ary  $n$ -tree, bidirectional Clos  $k$ -ary  $n$ -tree, and MiKANT( $k, n$ ). From the table, we know that the proposed MiKANT( $k, n$ ) has less switches, less links, and less average distance than other two networks for the same number of nodes in the networks.

### IV. ROUTING ALGORITHM FOR MiKANT

There are many routing algorithms for traditional  $k$ -ary  $n$ -trees. The basic idea is to find an NCA of both the source and destination nodes. In the phase of routing from the source node to the NCA, there are several paths. The adaptive routing algorithms only focus on the path selection in this phase. After that, the routing path from the NCA to the destination node is deterministic.

TABLE I  
COMPARISON OF TOPOLOGICAL PROPERTIES

	Classical $k$ -ary $n$ -tree	Clos $k$ -ary $n$ -tree	Mirrored $k$ -ary $n$ -tree
Number of nodes	$k^n$	$2k^n$	$2k^n$
Number of switches	$nk^{n-1}$	$(2n-1)k^{n-1}$	$(2n-2)k^{n-1}$
Number of links	$nk^n$	$2nk^n$	$(2n-1)k^n$
Radix	$2k$	$2k$	$2k$
Diameter	$2n$	$2n$	$2n$
Bisection	$k^n/2$	$k^n/2$	$k^n/2$
Average distance	$2n - \frac{2}{k-1} + \frac{2}{(k-1)k^n}$	$2n - \frac{1}{k-1} + \frac{1}{(k-1)k^n}$	$2n - \frac{1}{k-1} + \frac{1}{(k-1)k^n} - \frac{1}{2}$

In this section we give a minimal per-hop deterministic routing algorithm based on the ID definition of MiKANT( $k, n$ ). Our algorithm finds a routing path with the minimal number of switches. The output port in each switch is selected based on the IDs of the current switch and destination node. Suppose that a source node  $S$  is represented by  $S = \langle G_S, S_{n-1}, S_{n-2}, \dots, S_1, S_0 \rangle$ . It wants to send a *packet* to a destination node  $T$ , represented by  $T = \langle G_T, T_{n-1}, T_{n-2}, \dots, T_1, T_0 \rangle$ . First,  $S$  will send the packet to the switch  $\langle G_S, 0, S_{n-2}, \dots, S_1, S_0 \rangle$ , and finally,  $T$  will receive the packet from the switch  $\langle G_T, 0, T_{n-2}, \dots, T_1, T_0 \rangle$ .

In the phase of going to the NCA, a switch  $W$  in stage  $L_W$   $W = \langle G_W, L_W, W_{n-2}, \dots, W_{L_W+1}, W_{L_W}, T_{L_W-1}, \dots, T_0 \rangle$  sends the packet to a switch  $U$  which is closer to  $T$  than  $W$

$$U = \langle G_U, L_U, W_{n-2}, \dots, W_{L_W+1}, T_{L_W}, T_{L_W-1}, \dots, T_0 \rangle$$

where  $W_{L_W}$  is changed to  $T_{L_W}$ . That is, we change  $S_{n-2}, \dots, S_0$  as quickly to  $T_{n-2}, \dots, T_0$  as possible, in the field sequence of  $0, 1, \dots, n-2$ . In the phase of going to the destination switch from the NCA, the path is deterministic.

There are  $2k$  ports in a switch. We use a number in the range of  $0$  and  $2k-1$  as the port label (see Fig. 3). The port labels of  $k$  to  $2k-1$  are assigned for the phase of going to the NCA, and port labels of  $0$  to  $k-1$  are assigned for the phase of going to the destination node from the NCA. The switch controller should determine which port will be used for sending the packet, based on its switch ID and the destination information in the received packet.

The routing algorithm for each switch  $W = \langle G_W, L_W, W_{n-2}, \dots, W_1, W_0 \rangle$  is formally given in **Algorithm 1**, where  $T_{L_W}^+$  is the port label of switch  $W$  that is linked to a switch whose stage equals  $L_W + 1$  (increasing stage); its value equals the destination field  $T_{L_W} + k$ . Similarly,  $T_{L_W}^-$  is the port label of switch  $W$  that is linked to a switch whose stage equals  $L_W - 1$  (decreasing stage); its value equals the destination field  $T_{L_W}$  (see also Fig. 3).

Through  $T_{L_W}^+$  port, the *packet* is sent to the NCA of the source node and destination node. Through  $T_{L_W}^-$  port, the *packet* is sent toward destination node from the NCA. In this phase, because the fields  $W_{n-2}, \dots, W_1, W_0$  are already equal to  $T_{n-2}, \dots, T_1, T_0$ , the work is only to decrease the stage. In the last step, the *packet* is sent by the switch in stage  $0$  to the ultimate destination node through the  $T_{n-1}^-$  port. Note that

**Algorithm 1** MiKANT\_Routing (*packet*)

```

Input: packet =  $\langle T, data \rangle$ ; /* received packet which will be sent to  $T$  */
 $T = \langle G_T, T_{n-1}, T_{n-2}, \dots, T_1, T_0 \rangle$ ; /* destination node ID */
 $W = \langle G_W, L_W, W_{n-2}, \dots, W_1, W_0 \rangle$ ; /* my switch ID */
if ( $G_W \neq G_T$ ) /*  $W, T$ : different groups */
    send packet to  $T_{L_W}^+$  port; /* increasing stage */
else /*  $W, T$ : same group */
    if ( $W_{n-2}, \dots, W_0 \neq T_{n-2}, \dots, T_0$ ) /* going to NCA */
        send packet to  $T_{L_W}^+$  port; /* increasing stage */
    else /* going to destination from NCA */
        if ( $L_W \neq 0$ ) /* not a stage 0 switch */
            send packet to  $T_{L_W}^-$  port; /* decreasing stage */
        else /* a stage 0 switch */
            send packet to  $T_{n-1}^-$  port; /* to destination node */
        endif
    endif
endif

```

TABLE II  
ROUTING EXAMPLES IN MiKANT(3, 4)

	$G_T \neq G_S$	$G_T = G_S$	$G_T = G_S$	$G_T = G_S$
$S$	$\langle 0, 2, 0, 0, 0 \rangle$	$\langle 0, 2, 0, 0, 0 \rangle$	$\langle 0, 2, 0, 0, 0 \rangle$	$\langle 0, 2, 0, 0, 0 \rangle$
Switches	$\langle 0, 0, 0, 0, 0 \rangle$ 5	$\langle 0, 0, 0, 0, 0 \rangle$	$\langle 0, 0, 0, 0, 0 \rangle$	$\langle 0, 0, 0, 0, 0 \rangle$
	$\langle 0, 1, 0, 0, 2 \rangle$ 5	$\langle 0, 1, 0, 0, 2 \rangle$	$\langle 0, 1, 0, 0, 2 \rangle$	$\langle 0, 1, 0, 0, 2 \rangle$
	$\langle 0, 2, 0, 2, 2 \rangle$ 5	$\langle 0, 2, 0, 2, 2 \rangle$	$\langle 0, 2, 0, 2, 2 \rangle$	$\langle 0, 0, 0, 0, 2 \rangle$
	$\langle 1, 2, 2, 2, 2 \rangle$ 2	$\langle 1, 2, 2, 2, 2 \rangle$	$\langle 0, 1, 0, 2, 2 \rangle$	
	$\langle 1, 1, 2, 2, 2 \rangle$ 2	$\langle 0, 2, 2, 2, 2 \rangle$	$\langle 0, 0, 0, 2, 2 \rangle$	
	$\langle 1, 0, 2, 2, 2 \rangle$ 2	$\langle 0, 1, 2, 2, 2 \rangle$	$\langle 0, 0, 2, 2, 2 \rangle$	
$T$	$\langle 1, 2, 2, 2, 2 \rangle$	$\langle 0, 2, 2, 2, 2 \rangle$	$\langle 0, 2, 0, 2, 2 \rangle$	$\langle 0, 2, 0, 0, 2 \rangle$

$T_{L_W}^- \in \{0, 1, \dots, k-1\}$  and  $T_{L_W}^+ \in \{k, k+1, \dots, 2k-1\}$ . This minimal per-hop deterministic routing algorithm is quite simple and can be implemented with a simple fixed logic to determine the output port. It is deadlock free for the collective communications.

Table II shows four routing examples in MiKANT( $k, n$ ) with  $k=3$  and  $n=4$ , based on **Algorithm 1**.  $S$  and  $T$  are the source and destination nodes, respectively. The others in the center part are switch IDs. The first example shows the routing path between  $S$  and  $T$  whose groups are different. The number next to the switch label is the label of the used port (see Fig. 4). The path length is  $2n-1=7$ . The second example shows the routing path between  $S$  and  $T$  whose groups are the same. The path length equals the diameter which is  $2n=8$ . The other two examples show short paths in the case that  $S$  and  $T$  are in the same group.

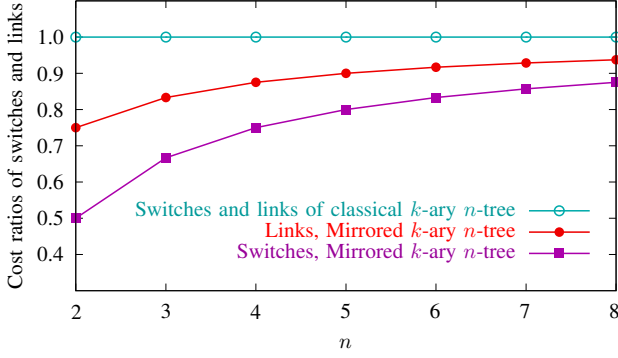


Fig. 5. Cost ratios of links and switches

## V. COST PERFORMANCE EVALUATION

This section examines the ratios of the switches and links which the MiKANT can reduce, evaluates the performance improvement compared to the classical  $k$ -ary  $n$ -tree, describes how to determine the values of  $k$  and  $n$  for a given node size  $N$ , and shows the average packet latencies through event simulations.

### A. Analytical Evaluation

The classical  $k$ -ary  $n$ -tree has  $k^n$  nodes and uses  $nk^{n-1}$  switches. On average, one node requires  $nk^{n-1}/k^n$  switches. MiKANT( $k, n$ ) has  $2k^n$  nodes and uses  $(2n-2)k^{n-1}$  switches. The switch cost ratio of MiKANT( $k, n$ ) to the classical  $k$ -ary  $n$ -tree is

$$R_S = \frac{(2n-2)k^{n-1}/(2k^n)}{nk^{n-1}/k^n} = 1 - \frac{1}{n} \quad (4)$$

Similarly, the link cost ratio of MiKANT( $k, n$ ) to the classical  $k$ -ary  $n$ -tree is

$$R_L = \frac{(2n-1)k^n/(2k^n)}{nk^n/k^n} = 1 - \frac{1}{2n} \quad (5)$$

Fig. 5 plots these ratios. When  $n=2$ , MiKANT( $k, n$ ) saves 50% switches and 25% links; when  $n=8$ , MiKANT( $k, n$ ) saves 12.5% switches and 6.25% links.

In most literature, as one of the network topological properties, the *cost* is simply defined as the product of the radix and diameter. Decreasing radix will reduce the hardware cost, and decreasing diameter will shorten the communication time and hence improve performance. We compare the cost performance of MiKANT( $k, n$ ) to classical  $k$ -ary  $n$ -tree under the condition of the same number of compute nodes for two networks.

MiKANT( $k_m, n$ ) has  $2k_m^n$  nodes; the radix is  $2k_m$ ; and the diameter is  $2n$ . Suppose that we build a classical  $k_c$ -ary  $n$ -tree that also has  $2k_m^n$  nodes under a given same  $n$ , then we have  $k_c^n = 2k_m^n$ , or  $k_c = 2^{1/n}k_m$ . The cost performance improvement is

$$S = \frac{2k_c \times 2n}{2k_m \times 2n} = 2^{1/n} \quad (6)$$

Fig. 6 illustrates the cost performance improvement of MiKANT( $k, n$ ) compared to the classical  $k$ -ary  $n$ -tree. When

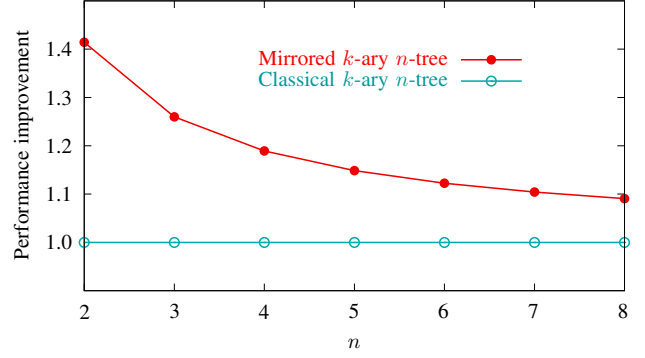


Fig. 6. Performance improvement of MiKANT( $k, n$ )

$n=2$ , MiKANT( $k, n$ ) improves performance of 41.4%; when  $n=8$ , MiKANT( $k, n$ ) improves performance of 9.1%.

If we want to build a MiKANT( $k, n$ ) with a given number of nodes  $N = 2k^n$ , there is a question about how to determine the values of  $k$  and  $n$ . Our goal is to enlarge the performance at low hardware cost. To do that, we define a *relative cost performance* (RCP) to the hypercube as following.

$$\text{RCP} = \frac{2k \times 2n}{(\log_2(2k^n/p) + p) \times (\log_2(2k^n/p) + 2)} \quad (7)$$

where  $2k$  is the MiKANT( $k, n$ ) switch radix which affects the hardware cost;  $2n$  is the diameter which affects the communication performance;  $\log_2(2k^n/p)$  is the dimension of the hypercube, and  $p$  is the number of ports in a router for connecting compute nodes.

When we talk about topological properties of an  $m$ -dimensional hypercube, or  $m$ -cube, we say that both its radix and diameter are  $m$ . But in real implementations, there are  $p$  ports in a router for connecting compute nodes. Therefore, the real router radix is  $m+p$ . Also, the diameter of MiKANT( $k, n$ ), which is  $2n$ , contains the links that connect compute nodes to switches. To make a fair comparison, we let the diameter of  $m$ -cube be  $m+2$ .

Fig. 7 plots the RCP of MiKANT( $k, n$ ) to hypercube for  $2 \leq n \leq 8$  with  $p=1$ . For a given  $n$ , we change  $k$  to implement the system with different sizes. The lower values in the curves means that the systems can be constructed with higher performance at lower hardware cost. If the value is less than 1, we say that it is better than the hypercube. We can see that each curve has a minimum RCP value at a horizontal position which is the size of the system. This figure can help us to determine the values of  $k$  and  $n$  for a given system scale. For example, when we build a 524,288-node system, we can let  $k=8$  and  $n=6$ ; the RCP is 0.457.

### B. Synthetic Simulation

We have evaluated the average packet latencies of MiKANT and bidirectional Clos network with  $k=4$  and  $n=5$  by the proposed routing algorithm through event simulation. Both networks have the same number of compute nodes which is  $2k^n = 2,048$ . The reason why select  $k=4$  and  $n=5$  is that



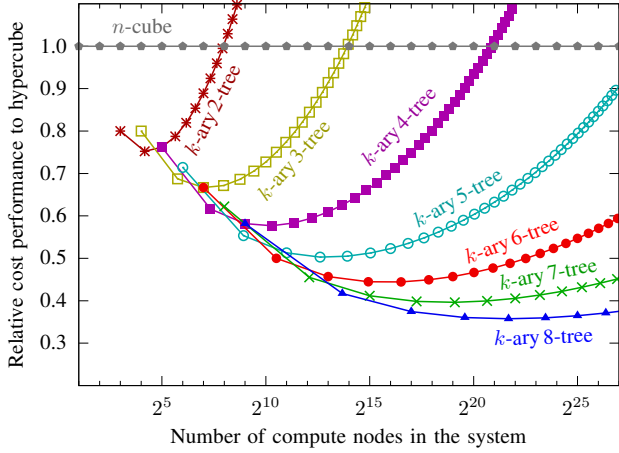


Fig. 7. RCP comparison of Mirrored  $k$ -ary  $n$ -tree

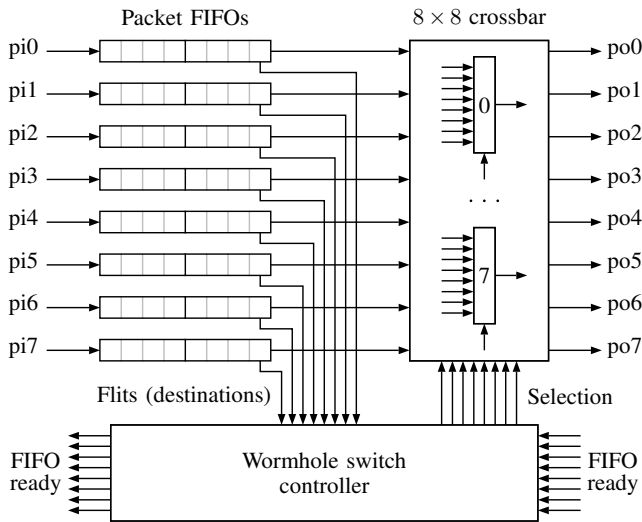


Fig. 8. Block diagram of switch for the simulation

the simulator needs to use a big amount of memory and we did it in an ordinary personal computer.

The block diagram of the switch for the simulation is shown in Fig. 8. There is an  $8 \times 8$  crossbar. The output port  $po_i$  and input port  $pi_i$  form a bidirectional channel  $i$  for  $0 \leq i \leq 7$ . Each input port of the switch has a FIFO (first-in first-out) buffer for queuing packets. The depth of the FIFO is two (packets). In addition to the channels for packet exchanging, there are eight bidirectional signals (“FIFO ready” signals in the figure) that inform its neighbor switches of the feasibility of the corresponding FIFO and get the FIFO state of the neighbor switches. We need such one-hop FIFO state information for sending packets to the next switch. Our switch can perform the wormhole switching routing. We let the flit length be 32 bits which can accommodate a destination node address. Because the minimal per-hop deterministic routing algorithm is quite simple, we assume that the switch controller can determine the output port for the packet in the head of the FIFO buffer in one

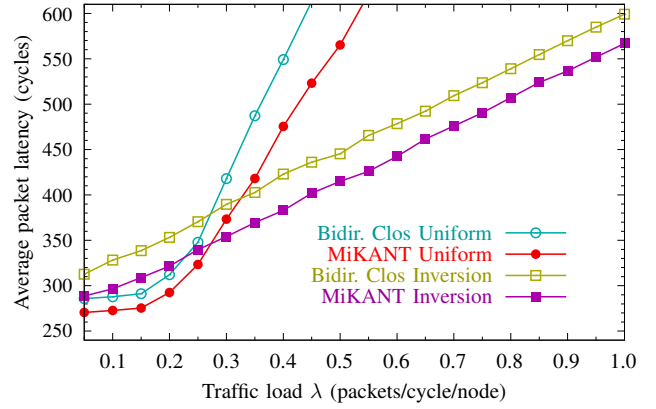


Fig. 9. 4-ary 5-tree average packet latencies

clock cycle once the address flit arrived (“Flits (destinations)” signals in the figure). In this simulation, the arbitration for each output port is performed in a random way, that is, on every clock cycle, one packet is selected randomly from multiple packets that are sent through the same output port (“Selection” signals in the figure).

We perform the simulation on a clock cycle-by-cycle basis and simulate two synthetic traffic patterns: uniform and bit-inversion. In uniform traffic, packet destination addresses are randomly assigned. In bit-inversion traffic, a compute node sends all its packets to the node whose address is the bit-inversion of the sending node address. We set traffic load  $\lambda$  in the range of 0.05 and 1.00, stepped by 0.05. This means that on every clock cycle, there are  $N \times \lambda$  compute nodes that send packets to their destination nodes where  $N$  is the number of compute nodes in the system. A congested network has a big value of  $\lambda$ . For example, in the all-to-all personalized communication, the  $\lambda$  equals 1.00. The simulation terminates when at least 200 packets of each source node reached their destinations. The calculation of the packet latency does not contain those packets more than 200 for each source node.

Fig. 9 shows the simulation results for the bidirectional Clos and Mirrored 4-ary 5-trees. The vertical axis represents the average packet latency in clock cycles and the horizontal axis represents the traffic load. Both networks have 2,048 compute nodes. MiKANT has  $(2 \times 5 - 2) \times 4^{5-1}$ , or 2,048, switches while bidirectional Clos network has  $(2 \times 5 - 1) \times 4^{5-1}$ , or 2,304, switches. We can see that MiKANT has lower average packet latencies than the bidirectional Clos network in both the traffic patterns.

From the figure, we understand that when the traffic load is low, the latency for the uniform pattern is less than that for the bit-inversion pattern. This is because the uniform traffic contains short paths between the source and destination nodes, and the conflicts on output ports are low. On the other hand, as the traffic load becomes heavier, the bit-inversion pattern gets lower latency than the uniform pattern, because the bit-inversion pattern has fewer conflicts on switch output ports than the uniform pattern, as shown as in Fig. 10. In the bit-

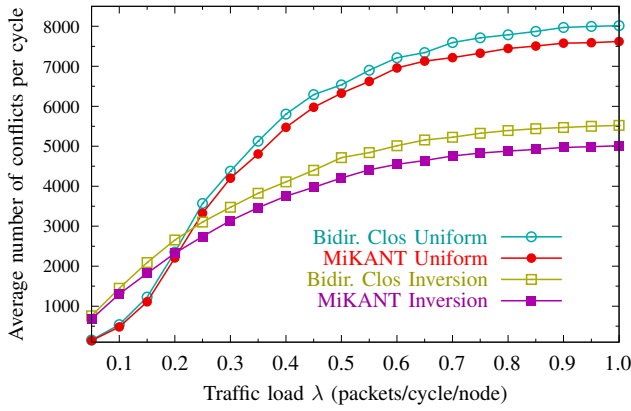


Fig. 10. 4-ary 5-tree average output port conflicts

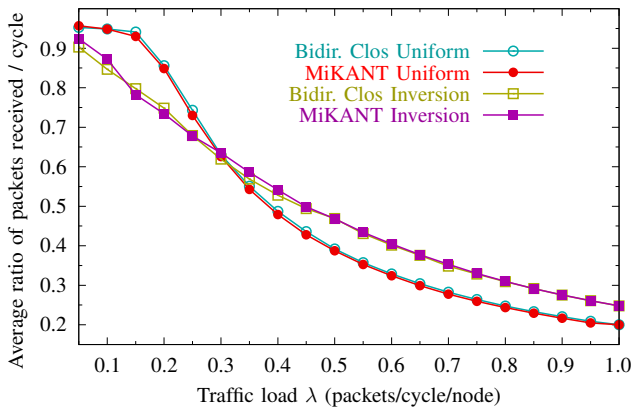


Fig. 11. 4-ary 5-tree ratio of packets received

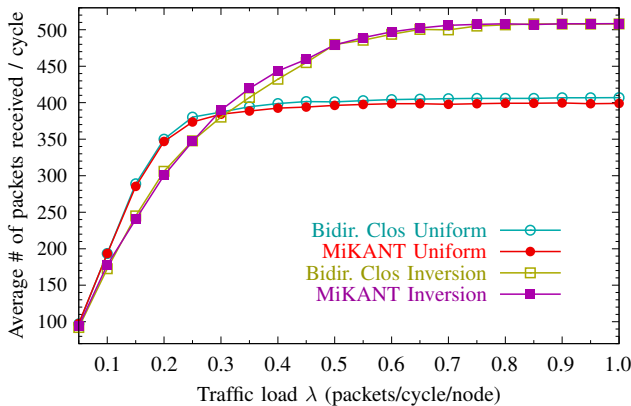


Fig. 12. 4-ary 5-tree average packets received

inversion pattern, all the paths have the same length and are well distributed.

Note that in our simulation, we ignored the signal propagation time on the links. Long link will result in long propagation time. Because MiKANT has less links and shorter average distance than the bidirectional Clos network, counting the propagation time will make the performance of MiKANT even better than the bidirectional Clos network.

Fig. 11 shows the ratio of the number of the packets received, including the packets more than 200, to the number of all the packets sent on every clock cycle. Fig. 12 shows the average number of the packets received, including the packets more than 200, per clock cycle. From these two figures, we know that MiKANT and bidirectional Clos network have no obvious difference on these two measures. This means that the both networks have almost the same communication capacities.

## VI. CONCLUSIONS

Recent years, many supercomputers, including the data centers, adopt the interconnection networks based on the fat-tree and Clos topologies. The proposed MiKANT has almost the same communication capacity as that of the classical fat-tree and bidirectional Clos networks but MiKANT uses fewer switches and links and has shorter average distance than those two networks. The results of the analytical evaluation and synthetic simulation show that MiKANT can reduce both the hardware cost and average packet latency. This means that MiKANT can achieve high communication performance at low implementation cost. The future research work may include the development of the adaptive routing algorithms for MiKANT, fault tolerant routing in MiKANT, and implementation of MiKANT on NoC.

## REFERENCES

- [1] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct. 1985.
- [2] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallel architectures," in *Proceedings 11th International Parallel Processing Symposium*, Apr. 1997, pp. 87–93.
- [3] Z. Qian, B. Hu, and K. L. Yeung, "An efficient routing algorithm in fat-tree data center networks," in *2016 IEEE Global Communications Conference*, Dec. 2016, pp. 1–6.
- [4] C. Clos, "A study of non-blocking switching networks," *The Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, Mar. 1953.
- [5] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Holzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *2015 ACM Conference on Special Interest Group on Data Communication*, Aug. 2015, pp. 183–197.
- [6] D. Abts and J. Kim, *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities*. Morgan and Claypool Publishers, Mar. 2011.
- [7] C. Gómez, F. Gilabert, M. E. Gómez, P. López, and J. Duato, "Ruft: Simplifying the fat-tree topology," in *14th IEEE International Conference on Parallel and Distributed Systems*, Dec. 2008, pp. 153–160.
- [8] 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, Apr. 2016.
- [9] D. Ludovici, F. Gilabert, C. Gómez, M. E. Gómez, P. López, G. Gaydadjiev, and J. Duato, "Butterfly vs. unidirectional fat-trees for networks-on-chip: not a mere permutation of outputs," in *3rd Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip, in conjunction with HiPEAC Conference*, Jan. 2009.
- [10] 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, June 2014.
- [11] J. Navaridas, J. Miguel-Alonso, F. J. Ridruejo, and W. Denzel, "Reducing complexity in tree-like computer interconnection networks," *Parallel Computing*, vol. 36, no. 2-3, pp. 71–85, Feb.-Mar 2010.