International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 11, Number 2, pages 140–153, July 2021

Link Fault Tolerant Routing Algorithms in Mirrored K-Ary N-Tree Interconnection Networks

Yaodong Wang
Graduate School of CIS
Hosei University
yaodong.wang.7y@stu.hosei.ac.jp

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

Received: February 12, 2021
Revised: April 28, 2021
Accepted: May 27, 2021
Communicated by Eitaro Kohno

Abstract

This paper investigates the fault tolerance of Mirrored k-Ary n-Tree (MiKANT) networks with link faulty. The MiKANT network is a variant of the traditional k-ary n-tree (Fat-tree) and Clos networks. It doubles the number of compute nodes of the fat-tree by adding a few switches and links and has a shorter average distance to reduce the packet latency. As the scale of MiKANT becomes larger, the probability of link faulty becomes higher. In order to improve the successful routing ratio of MiKANT, we give four link fault tolerant routing algorithms for MiKANT and evaluate their performance through simulations. In addition, the performance of the combined algorithms is also evaluated.

Keywords: interconnection network, fat-tree, mirrored k-ary n-tree, routing, fault tolerance, algorithm, simulation

1 Introduction

The fat-tree [4] is one of the most popular topologies of the interconnection networks in current large-scale supercomputers. In a k-ary n-tree [9], one of the fat-trees, the switch radix is $2k$, and the number of levels is n. The compute nodes are connected only to the leaf switches. Fat-tree networks provide the non-blocking feature with 1 : 1 oversubscription ratio and high path diversity but meanwhile require a great number of switches and links with a complex wire connection, and hence increase the hardware cost and packet latency. [7] and [10] focused on the cost of the switches and links, but their proposed topologies reduce the diversity of paths.

In order to reduce the hardware cost of the switches and links and improve the communication performance of the fat-tree networks, a Mirrored K-Ary N-Tree (MiKANT) network [5] was proposed. It is a variant of fat-trees aimed at reducing hardware cost and packet latency. As the scale of MiKANT becomes large, the probability of switch or link failure increases. Fault tolerance is one of the important issues of the interconnection networks for large-scale parallel computers. The dynamic fault tolerance in fat-trees was discussed in [11]; [6] has discussed switch fault tolerance in MiKANT. [12] proposed a new heuristic probe selection algorithm based on adaptive probing to solve consumed large number of probes and wavelengths in large-size networks. [8] investigated six Traffic Patterns and analyzed two Adaptive Routing Algorithms namely the Non-Minimal Adaptive Routing Algorithm and Minimal Adaptive Routing Algorithm. [1] proposed a further extension to the fat trees called Zoned-Fat tree, and gave a fault tolerance algorithm for it. [2] presented a fast deterministic routing algorithm for fat-trees, which minimizes congestion risk even under massive topology degradation caused by equipment failure.

This paper focuses on the link fault tolerance for MiKANT networks. The main contributions are to propose fault tolerant routing algorithms for MiKANT networks with link faulty and evaluate their performance through simulations. We also give a method to avoid the deadlock during the link fault tolerant routing. The rest of the paper is organized as follows. Section 2 introduces MiKANT networks. Section 3 gives link fault tolerant algorithms. Section 4 evaluates the performance of the algorithms. And Section 5 concludes the paper.

2 Mirrored K-Ary N-Tree

This section introduces the structure of the MiKANT network and its shortest path routing algorithm.

2.1 MiKANT Topology

A MiKANT(k, n) has two groups and each group has $n-1$ levels. There are $2(n-1)k^{n-1}$ switches, $(2n-1)k^n$ links, and $2k^n$ compute nodes. Each switch has $2k$ bidirectional ports. 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.

Each switch in a MiKANT(k, n) for $n \geq 2$ is labeled as

$$\langle G, L, D_{n-2}, D_{n-3}, \ldots, D_1, D_0 \rangle$$

each link in a MiKANT(k, n) for $n \geq 2$ is labeled as

$$\langle G, L, D_{n-2}, D_{n-3}, \ldots, D_1, D_0, P \rangle$$

and each compute node in a MiKANT(k, n) is labeled as

$$\langle G, C_{n-1}, C_{n-2}, C_{n-3}, \ldots, C_1, C_0 \rangle$$

where G indicates the *group* with $G \in \{0, 1\}$, L indicates the *level* with $L \in \{0, \ldots, n-2\}$, P indicates the *port* number of a switch with $P \in \{0, \ldots, k-1\}$, $C_{n-1}, C_{n-2}, \ldots, C_1, C_0$ is an n-tuple $\{0, 1, \ldots, k-1\}^n$, and $D_{n-2}, D_{n-3}, \ldots, D_1, D_0$ is an $(n-1)$-tuple $\{0, 1, \ldots, k-1\}^{n-1}$ which identifies the switch inside level L of group G. A switch

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

connects to switches

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

via links

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

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

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

via links

$$\langle 1, L, *, D_{n-3}, \ldots, D_1, D_0, P \rangle$$

with $P = D_{L-1}$, $\overline{G}$ is the bit-inversion of G. $*$ is any value for $* \in \{0, 1, \ldots, k-1\}$. Note that the links in between two groups are labeled with $G = 1$ and $L = n - 1$. For example, a level 1 switch of group 0 $W = \langle 0, 1, 0, 0 \rangle$ in a MiKANT(3, 3) ($k = 3$ and $n = 3$) connects to switches $\langle 1, 1, 0, 0 \rangle$, $\langle 1, 1, 1, 0 \rangle$, and $\langle 1, 1, 2, 0 \rangle$ via links $\langle 1, 2, 0, 0, 0 \rangle$, $\langle 1, 2, 1, 0, 0 \rangle$, and $\langle 1, 2, 2, 0, 0 \rangle$, with $P = D_{L-1} = 0$. The switch $W = \langle 0, 1, 0, 0 \rangle$ also connects to switches $\langle 0, 0, 0, 0 \rangle$, $\langle 0, 0, 0, 1 \rangle$, and $\langle 0, 0, 0, 2 \rangle$ via links $\langle 0, 1, 0, 0, 0 \rangle$, $\langle 0, 1, 0, 0, 1 \rangle$, and $\langle 0, 1, 0, 0, 2 \rangle$, with $P = D_{L-1} = *$. A compute node

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

connects to the switch

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

via link

$$\langle G, 0, D_{n-2}, \ldots, D_1, D_0, C_{n-1} \rangle$$

For example, a level 0 switch of group 0 $W = \langle 0, 0, 0, 0 \rangle$ in a MiKANT(3, 3) connects to nodes $\langle 0, 0, 0, 0 \rangle$, $\langle 0, 1, 0, 0 \rangle$, and $\langle 0, 2, 0, 0 \rangle$ via links $\langle 0, 0, 0, 0, 0 \rangle$, $\langle 0, 0, 0, 0, 1 \rangle$, and $\langle 0, 0, 0, 0, 2 \rangle$.

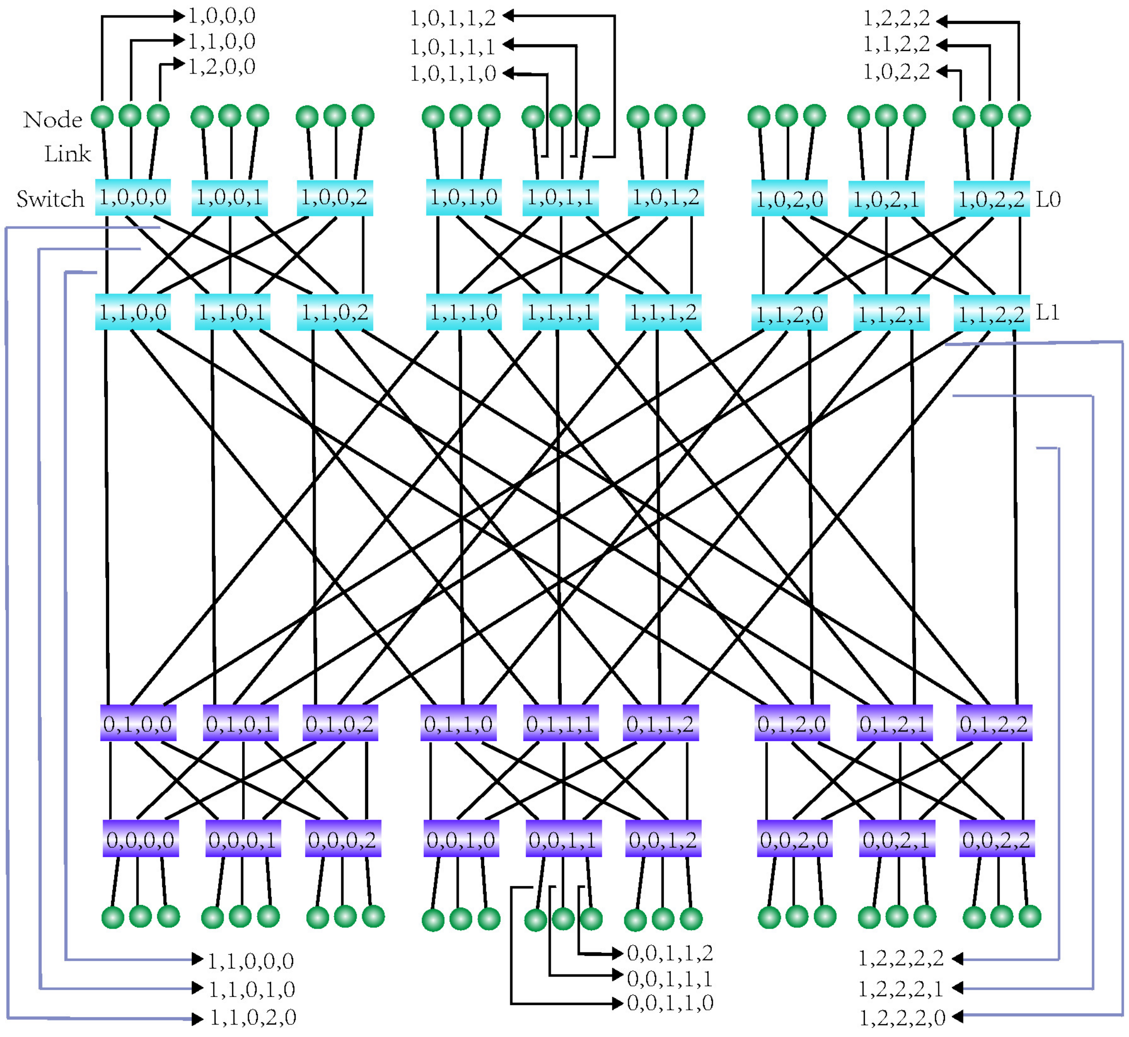

Figure 1: A Mirrored 3-ary 3-tree

Fig. 1 shows a MiKANT(3, 3). There are $2(n-1) = 4$ levels and each level has $k^{n-1} = 9$ switches, and the total number of the switches is $2(n-1)k^{n-1} = 36$. The number of compute nodes

Algorithm 1 MiKANT_Routing (*packet*)

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

is $2k^n = 54$, and the number of links is $(2n-1)k^n = 135$. The upper two levels are of group 1 and the lower two levels are of group 0. The figure also shows the link labels, switch labels, and node labels.

We summarize the topological properties of MiKANT(k, n) as follows. The radix of all the switches is $2k$. The diameter of MiKANT(k, n) is $2n$. The bisection width of MiKANT(k, n) is $k^n/2$. The average distance of MiKANT(k, n) is $(2n-1)/(k-1) + 1/(k-1)k^n - 1/2$.

2.2 Routing Algorithm

The routing algorithm is based on the destination compute node ID and switch IDs. The output port on each switch is selected based on the current switch ID and destination node ID. Each switch has $2k$ ports in a MiKANT(k, n). In each group, the ports in the side near to compute nodes are labeled with $0, 1, \ldots, k-1$; the ports in the other side are labeled with $k, k+1, \ldots, 2k-1$. We call the switch connected to the source (destination) compute node source (destination) switch.

We use $W = \langle G_W, L_W, W_{n-2}, ..., W_1, W_0 \rangle$ to denote the current switch ID. The packet received by W contains the destination node ID $T = \langle G_T, T_{n-1}, T_{n-2}, ..., T_1, T_0 \rangle$. Based on T, W selects a port and sends the packet through the selected port. **Case 0**: If $G_W \neq G_T$ (W and T are of different groups), W sends the packet to an upper-level switch through the port $T_{L_W} + k$. For example, in a MiKANT$(3, 3)$, for $W = \langle 1, 0, 0, 1 \rangle$ and $T = \langle 0, 2, 1, 2 \rangle$, W sends the packet through the port $T_0 + k = 2 + 3 = 5$, to the switch $\langle 1, 1, 0, 2 \rangle$. Similarly, switch $\langle 1, 1, 0, 2 \rangle$ sends the packet through the port $T_1 + k = 1 + 3 = 4$, to the switch $\langle 0, 1, 1, 2 \rangle$. These routings are in an *upward phase* in which the packet is sent from the source switch to a nearest common ancestor (NCA) of both the source and destination nodes. Because there are more than one NCA, we can select other ports to send the packet to different NCAs. This is helpful for the fault tolerant routing. If $G_W = G_T$ (W and T are of the same group), there are three cases. **Case 1**: $W_{n-2}, ..., W_{L_W} = T_{n-2}, ..., T_{L_W}$ and $L_W > 0$. This means that the packet already reached the NCA and the routing enters to a *downward phase*. In this phase, the routing is deterministic. The port of $T_{L_W - 1}$ must be selected for sending the packet toward the destination switch. For example, in a MiKANT$(3, 3)$, for $W = \langle 0, 1, 1, 2 \rangle$ and $T = \langle 0, 2, 1, 2 \rangle$, W sends the packet through the port $T_{1-1} = 2$, to the switch $\langle 0, 0, 1, 2 \rangle$. **Case 2**: $W_{n-2}, ..., W_{L_W} = T_{n-2}, ..., T_{L_W}$ and $L_W = 0$. This means that the packet already reached the destination switch. The port of T_{n-1} will be selected for sending the packet to the destination compute node. For example, in a MiKANT$(3, 3)$, for $W = \langle 0, 0, 1, 2 \rangle$ and $T = \langle 0, 2, 1, 2 \rangle$, W sends the packet through the port $T_2 = 2$ to T. **Case 3**: $W_{n-2}, ..., W_{L_W} \neq T_{n-2}, ..., T_{L_W}$. This means that the packet did not yet reach the NCA. W will send the packet to an upper level switch through the port $T_{L_W} + k$. For example, in a MiKANT(3, 3), for $W = \langle 0, 0, 0, 0\rangle$ and $T = \langle 0, 2, 1, 2\rangle$, W sends the packet through the port $T_{L_W} + k = T_0 + k = 2 + 3 = 5$, to the switch $\langle 0, 1, 0, 2\rangle$. The routing algorithm is formally given in **Algorithm 1**, where $T_x^+ = T_x + k$ and $T_x^- = T_x$.

3 Link Fault Tolerance in MiKANT

The purpose of fault tolerant routing is to enable a system to continue operating properly in a high probability with switch or link faulty. A link faulty means that the ports of the switches connected by the link cannot be used for passing the packet. In this section, we give four link fault tolerant routing algorithms and their combinations. In addition, some algorithms are compared with other existing methods.

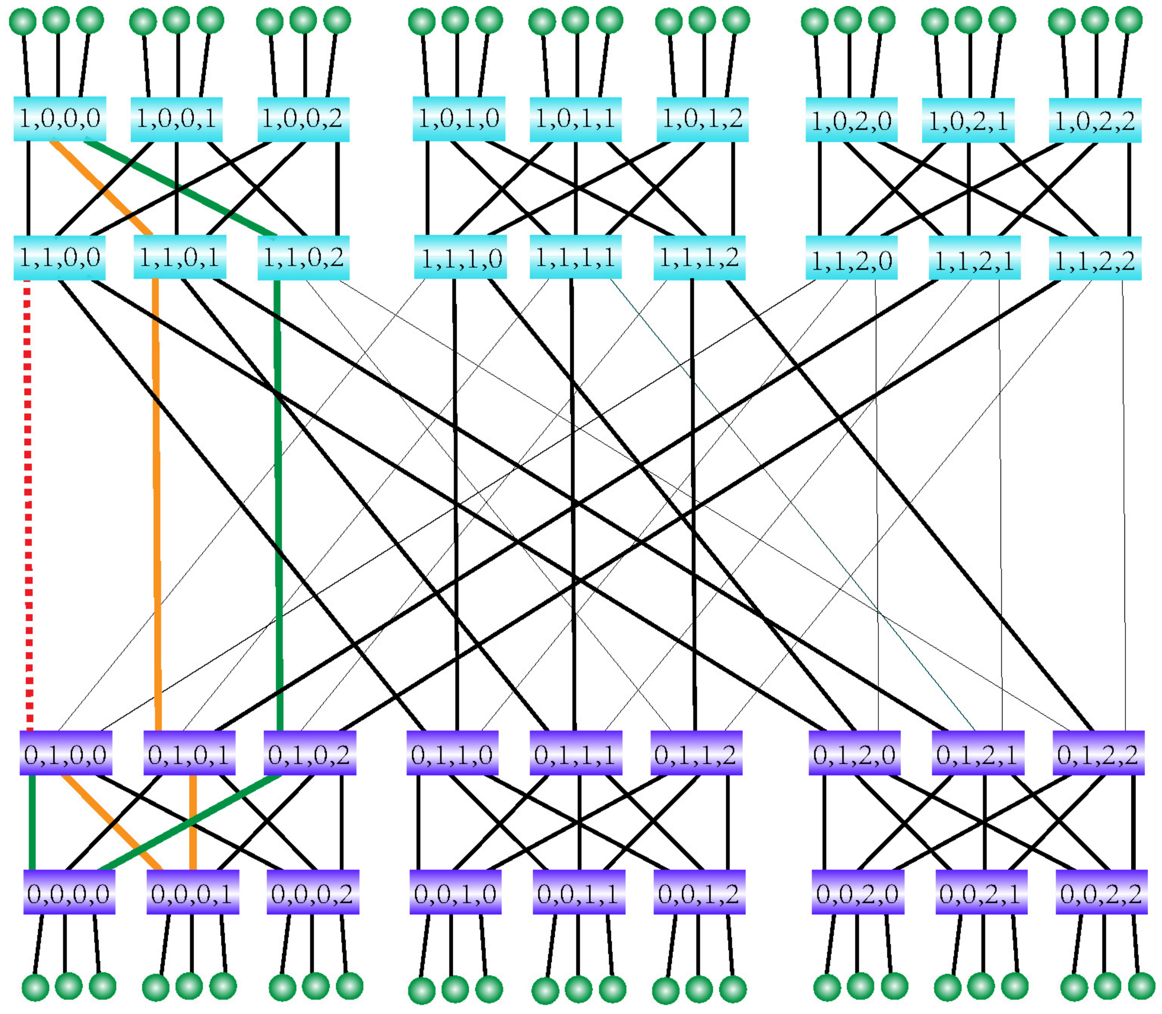

Figure 2: Go-Neighbor-Switch algorithm

3.1 Fault Tolerant Routing Algorithms

The routing algorithm given in **Algorithm 1** finds a shortest path between the source and destination compute nodes. If a link in the path is faulty while the path to the destination node is determined, the shortest path routing algorithm will fail. For example, the current switch is $\langle 0, 1, 0, 0\rangle$ in a MiKANT(3, 3) and the destination node is $\langle 1, 0, 0, 0\rangle$. In this case, if the link $\langle 1, 2, 0, 0, 0\rangle$ is faulty, the current switch cannot reach the switch $\langle 1, 1, 0, 0\rangle$ in the shortest path. Based on the shortest path routing algorithm, we give four link fault tolerant routing algorithms that select other links for the routing if a faulty link is encountered.

3.1.1 Go-Neighbor-Switch Algorithm

In the previous example, the link $\langle 1, 2, 0, 0, 0\rangle$ is faulty. We can send the packet to the neighbor switch $\langle 1, 1, 0, 1\rangle$ or $\langle 1, 1, 0, 2\rangle$ of switch $\langle 1, 1, 0, 0\rangle$. In this way, the current switch can send the packet to the destination node $\langle 1, 0, 0, 0\rangle$ via the lower level switch $\langle 0, 1, 0, 1\rangle$ or $\langle 0, 1, 0, 2\rangle$. We call this the "Go-Neighbor-Switch" algorithm. The current switch can go to the lower level and back to the current level to change to the neighbor switch, as shown as in the Fig. 2. The red dotted link is the faulty link and the bold links are the related links to the algorithm. The current switch is $\langle 0, 1, 0, 0\rangle$ and destination node is $\langle 1, 0, 0, 0\rangle$. We can select the path like $\langle 0, 1, 0, 0\rangle \rightarrow \langle 0, 0, 0, 1\rangle \rightarrow \langle 0, 1, 0, 1\rangle \rightarrow \langle 1, 1, 0, 1\rangle \rightarrow \langle 1, 0, 0, 0\rangle$ or $\langle 0, 1, 0, 0\rangle \rightarrow \langle 0, 0, 0, 0\rangle \rightarrow \langle 0, 1, 0, 2\rangle \rightarrow \langle 1, 1, 0, 2\rangle \rightarrow \langle 1, 0, 0, 0\rangle$. Generally, for $W = \langle G_W, L_W, W_{n-2}, \ldots, W_1, W_0\rangle$ and $T = \langle \overline{G}_W, T_{n-1}, T_{n-2}, \ldots, T_1, T_0\rangle$, suppose that the link $\langle 1, L_W + 1, L_{n-2}, \ldots, L_1, L_0, P\rangle$ is faulty. If $G_W = 1$, P equals T_{n-2} and $(L_{n-2}, \ldots, L_1, L_0 = W_{n-2}, \ldots, W_1, W_0)$. Otherwise, P equals W_{n-2} and $(L_{n-2}, \ldots, L_1, L_0 = T_{n-2}, \ldots, T_1, W_0)$. W will send a packet to $\langle G_W, L_W - 1, W_{n-2}, *, \ldots, W_1, W_0\rangle$, then to $\langle G_W, L_W, W_{n-2}, * - W_{n-3}, \ldots, W_1, W_0\rangle$, and next to $\langle \overline{G}_W, L_W, T_{n-2}, * - W_{n-3}, \ldots, W_1, W_0\rangle$, where $* - W_{n-3}$ are all the element $\in \{0, 1, \ldots, k-1\}$ except W_{n-3}.

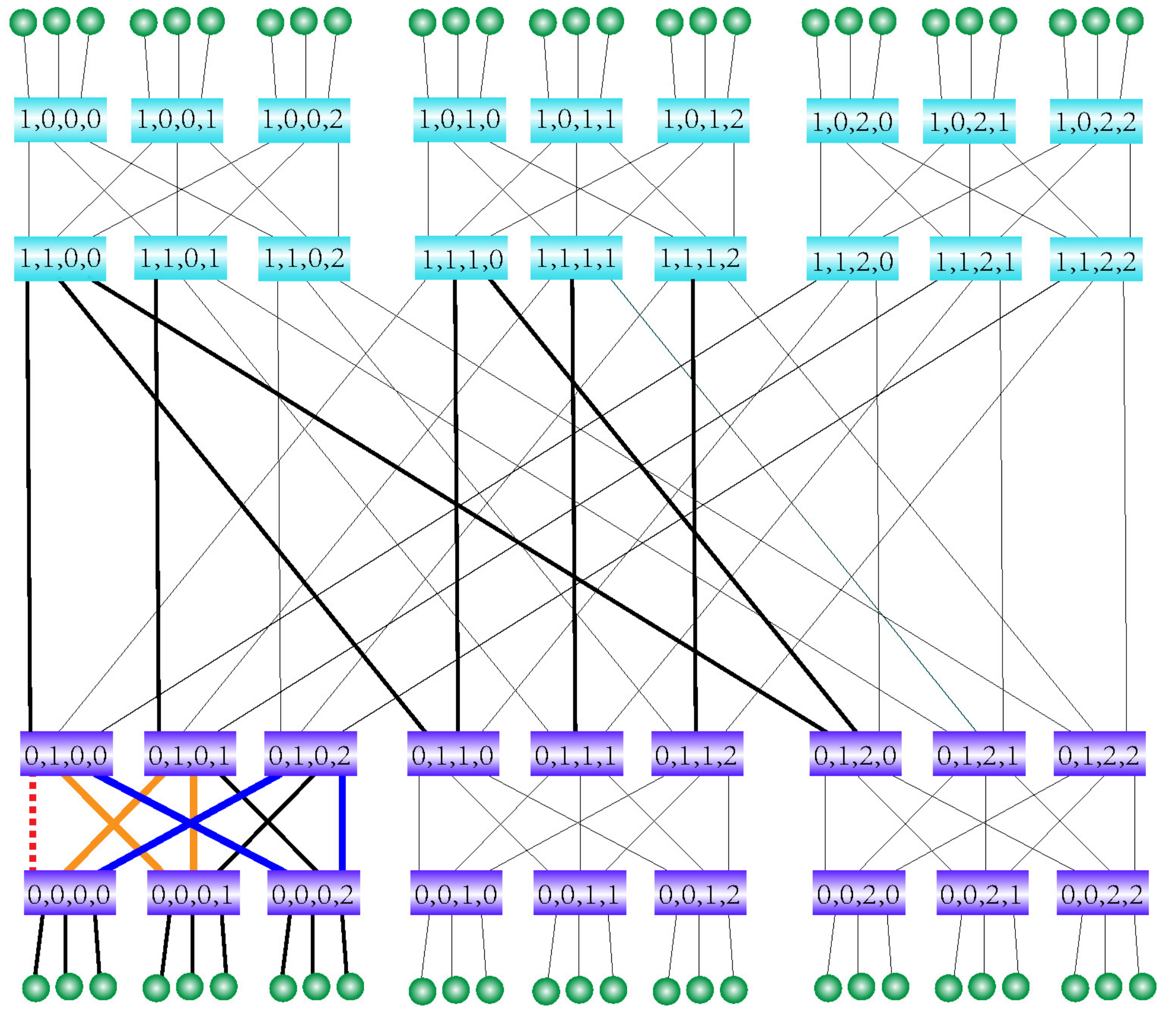

Figure 3: Go-Down-Level algorithm

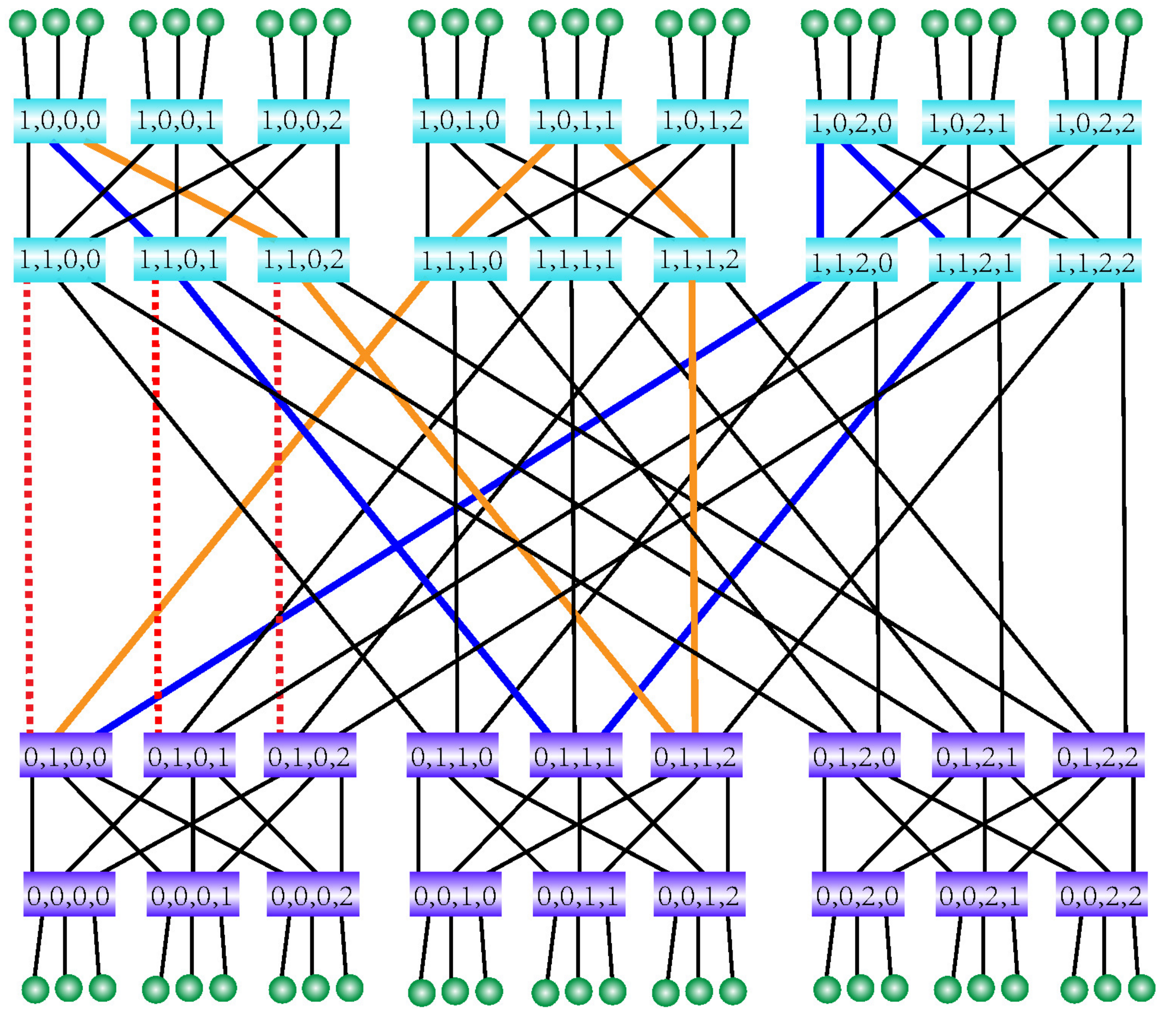

Figure 4: X-Turns algorithm

3.1.2 Go-Down-Level Algorithm

In the downward phase, the path to the destination node is determined. As shown as in the Fig. 3, the current switch $W = \langle 0, 1, 0, 0\rangle$ in a MiKANT(3, 3) wants to send a packet to the destination node $\langle 0, 2, 0, 0\rangle$ via switch $\langle 0, 0, 0, 0\rangle$. If the link $\langle 0, 1, 0, 0, 0\rangle$ is faulty, the shortest path algorithm will fail. But W can use other ways to get to the switch $\langle 0, 0, 0, 0\rangle$. For example, $\langle 0, 1, 0, 0\rangle \rightarrow \langle 0, 0, 0, 1\rangle \rightarrow \langle 0, 1, 0, 1\rangle \rightarrow \langle 0, 0, 0, 0\rangle$ or $\langle 0, 1, 0, 0\rangle \rightarrow \langle 0, 0, 0, 2\rangle \rightarrow \langle 0, 1, 0, 2\rangle \rightarrow \langle 0, 0, 0, 0\rangle$. We call this "Go-Down-Level" algorithm. Generally, for $W = \langle G_W, L_W,\ T_{n-2}, \ldots, T_1, *\rangle$, $T = \langle G_T, *, T_{n-2}, \ldots, T_1, T_0\rangle$, $G_W = G_T$, and $L_W = 1$, if $\langle G_W, L_W, T_{n-2}, \ldots, T_1, *, T_0\rangle$ is a faulty link, W can send the packet to the $\langle G_W, 0, T_{n-2}, \ldots, * - T_0\rangle$, then to $\langle G_W, 1, T_{n-2}, \ldots, * - W_0\rangle$, next to $\langle G_W, 0, T_{n-2}, \ldots, T_1, T_0\rangle$, and finally to the destination node.

3.1.3 X-Turns Algorithm

In the Go-Neighbor-Switch algorithm, when all the links connected to neighbor switches are faulty, W cannot send the packet to the destination node via neighbor switches. Referring to Fig. 4, switch $\langle 0, 0, 0, 0\rangle$ can not use Go-Neighbor-Switch algorithm to send packet to switch $\langle 1, 0, 0, 0\rangle$. But there are still other ways to get to the destination node. For example, $W = \langle 0, 1, 0, 0\rangle$ in a MiKANT(3, 3) routing to the destination node $\langle 1, 2, 0, 0\rangle$ and links $\langle 1, 2, 0, 0, 0\rangle$, $\langle 1, 2, 0, 1, 0\rangle$ and $\langle 1, 2, 0, 2, 0\rangle$ are faulty. We can send the packet to the destination node via $\langle 1, 1, 0, 1\rangle$ or

$\langle 1,1,0,2\rangle$ instead of the $\langle 1,1,0,0\rangle$, and we can route to the $\langle 1,1,0,1\rangle$ or $\langle 1,1,0,2\rangle$ by these ways: $\langle 0,1,0,0\rangle \to \langle 1,1,1,0\rangle \to \langle 1,0,1,1\rangle \to \langle 1,1,1,2\rangle \to \langle 0,1,1,2\rangle \to \langle 1,1,0,2\rangle \to \langle 1,0,0,0\rangle$ or $\langle 0,1,0,0\rangle \to \langle 1,1,2,0\rangle \to \langle 1,0,2,0\rangle \to \langle 1,1,2,1\rangle \to \langle 0,1,1,1\rangle \to \langle 1,1,0,1\rangle \to \langle 1,0,0,0\rangle$. We call this "X-Turns" algorithm. Generally, for $W = \langle G_W, L_W, W_{n-2}, \ldots, W_1, W_0\rangle$ and destination node $T = \langle \overline{G}_W, T_{n-1}, T_{n-2}, \ldots, T_1, T_0\rangle$. $\langle 1, L_W + 1, L_{n-2}, \ldots, L_1, L_0, P\rangle$ are faulty links. If $G_W = 1$, $(L_{n-2}, \ldots, L_1, L_0 = W_{n-2}, \ldots, W_1, *)$, P will equal T_{n-2}. Otherwise P will equal W_{n-2}. And $(L_{n-2}, \ldots, L_{L_W}, \ldots L_1, L_0 = W_{n-2}, \ldots, *, \ldots, T_1, T_0)$. The current switch can send a packet to $\langle \overline{G}_W, L^+_W, *-T_{n-2}, \ldots, W_1, W_0\rangle$, then to $\langle \overline{G}_W, L^+_W, *-T_{n-2}, \ldots, W_1, *-W_0\rangle$, next to $\langle G_W, L^+_W, *-W_{n-2}, \ldots, W_1, *-W_0\rangle$, and final to the destination node via switch $\langle \overline{G}_W, L^+_W, T_{n-2}, \ldots, W_1, *-W_0\rangle$ where $L^+_W = n-1$.

Algorithm 2 Go-Neighbor-Switch_and_X-Turns

```
$W = \langle G_W, L_W, W_{n-2}, ..., W_1, W_0\rangle$;                /* my switch ID */
$T = \langle G_T, T_{n-1}, T_{n-2}, ..., T_1, T_0\rangle$;            /* destination node ID */
$L = \langle G_L, L_L, L_{n-2}, ..., L_1, L_0, P\rangle$;             /* link ID */
if ($G_W \neq G_T$)                                  /* $W, T$: different groups */
    send packet to $T^+_{L_W}$ via link $L$;               /* increasing level */
    if ($L$ is faulty link)
        send packet to $W_N$ via link $L$;             /* Go-Neighbor-Switch */
        if ($L$ is faulty link)
            send packet to $N(T^+_{L_W-(n-2)})$ via link $L$;     /* X-Turns */
            if ($L$ is faulty link)
                routing fails;
            endif
        else
            if (current switch in $N(T^+_{L_W-(n-2)})$
                send packet to $\overline{T}^+_{L_W}$ via link $L$;       /* Back to */
                if ($L$ is faulty link)
                    routing fails;
                endif
            else
                if (current switch in $\overline{T}^+_{L_W}$)
                    send packet to $T^+_{L_W}$ via link $L$;
                    if ($L$ is faulty link)
                        routing fails;
                    endif
                endif
            endif
        endif                                        /* X-Turns end */
    else
        if (current switch in $W_N$)
            send packet to $T^+_{L_W}$ via link $L$;       /* Go to $T^+_{L_W}$ via $W_N$ */
            if ($L$ is faulty link)
                routing fails;
            endif
        endif
    endif
endif
```

X-Turns algorithm must be used together with Go-Neighbor-Switch algorithm, see **Algorithm 2**, where W_N means the neighbor switch of W, $N(T^+_{LW-(n-2)})$ means W has the same group ID as T and in level $L_W - (n-2) = 0$.

X-Turns algorithm is quite complex and may cause deadlock. In order to avoid the deadlock, we prepare two parameters to save the values of W_{n-2} and W_0 for each packet, and avoid selecting those paths that were visited before when the packet goes back to the switch.

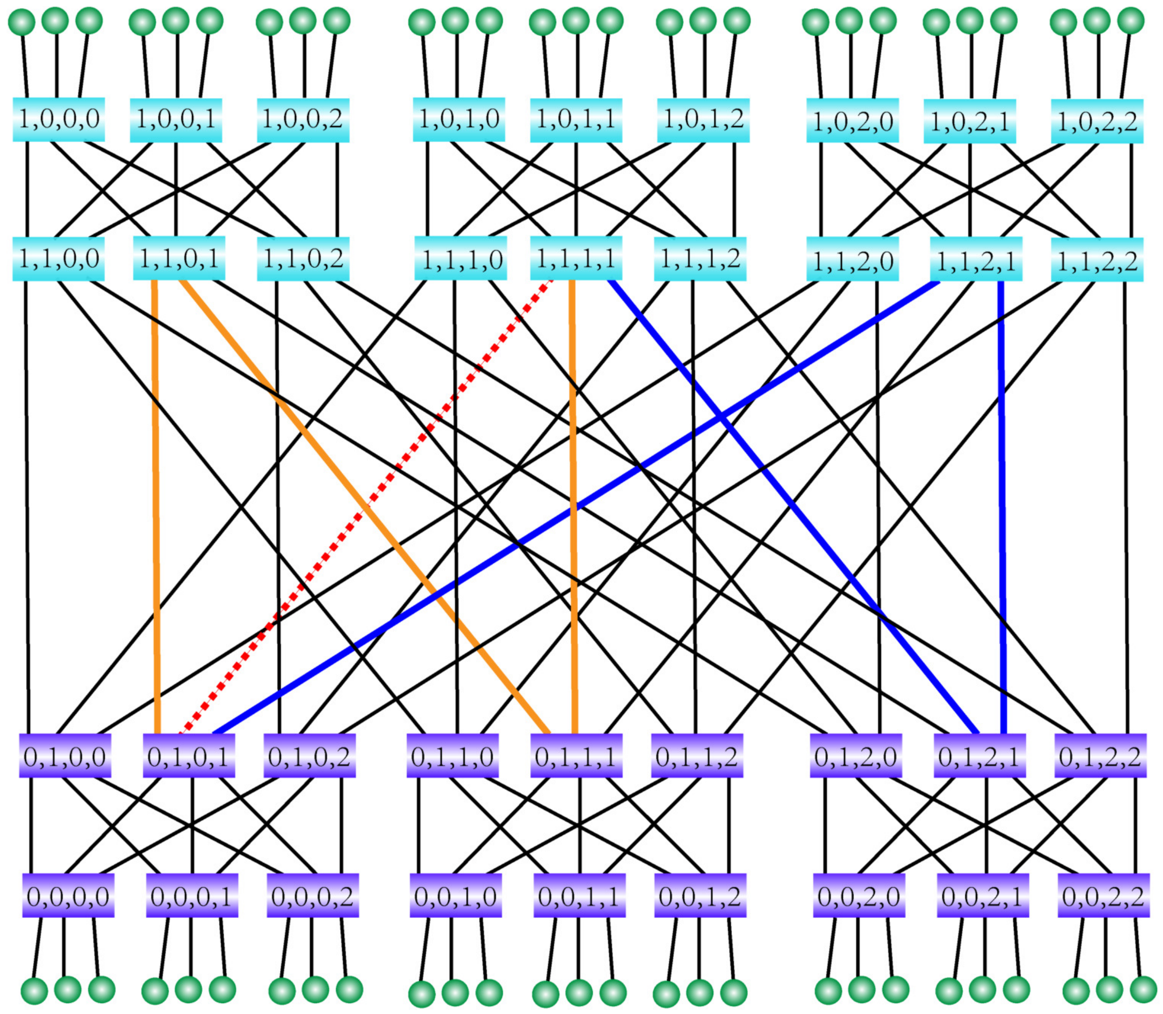

Figure 5: Three-Turn algorithm

3.1.4 Three-Turn Algorithm

As shown as in the Fig. 5. For $W = \langle 0,1,0,1\rangle$, in the shortest path algorithm, if the link $\langle 1,2,1,1,0\rangle$ is faulty, the packet cannot be sent to the destination node $\langle 1,0,1,1\rangle$ via this link. Another algorithm can be used by the following path, $\langle 0,1,0,1\rangle \rightarrow \langle 1,1,0,1\rangle \rightarrow \langle 0,1,1,1\rangle \rightarrow \langle 1,1,1,1\rangle$ or $\langle 0,1,0,1\rangle \rightarrow \langle 1,1,2,1\rangle \rightarrow \langle 0,1,2,0\rangle \rightarrow \langle 1,1,1,1\rangle$. It changes group three times, so we call it "Three-Turn" algorithm. Generally, for $W = \langle G_W, L_W, W_{n-2}, \ldots, W_1, W_0\rangle$ and destination node $T = \langle \overline{G}_W, T_{n-1}, T_{n-2}, \ldots, T_1, T_0\rangle$, suppose that $\langle 1, L_W + 1, L_{n-2}, \ldots, L_1, L_0, P\rangle$ is faulty. If $G_W = 1$, P equals T_{n-2} and $(L_{n-2}, \ldots, L_1, L_0 = W_{n-2}, \ldots, W_1, W_0)$. Otherwise, P equals W_{n-2} and $(L_{n-2}, \ldots, L_1, L_0 = T_{n-2}, \ldots, T_1, W_0)$. W will send a packet to $\langle \overline{G}_W, L_W, *{-}T_{n-2}, \ldots, W_1, W_0\rangle$, then to $\langle G_W, L_W, * - W_{n-2}, \ldots, W_1, W_0\rangle$, and next to $\langle \overline{G}_W, L_W, T_{n-2}, \ldots, W_1, W_0\rangle$.

In order to avoid the deadlock, we need a parameter to save the value of the W_{n-2} before W sends a packet to $\langle \overline{G}_W, L_W, *{-}T_{n-2}, \ldots, W_1, W_0\rangle$, and avoid selecting switch $\langle G_W, L_W, W_{n-2}, \ldots, W_1, W_0\rangle$ when the packet goes back from $\langle \overline{G}_W, L_W, * - T_{n-2}, \ldots, W_1, W_0\rangle$.

3.2 Combined Algorithms

In the previous sections, we described the Go-Neighbor-Switch, Go-Down-Level, X-Turns, and Three-Turn algorithms. Some algorithms improve performance when used in combination, and some algorithms compete with each other when used in combination. In this section, we give three combined algorithms that achieve better performance in the MiKANT.

3.2.1 Go-Neighbor-Switch and Go-Down-Level Algorithms

The Go-Neighbor-Switch algorithm is used when the current switch and destination/NCA are in different groups (upward phase), and the Go-Down-Level algorithm is used in the downward phase. Therefore, using the Go-Neighbor-Switch algorithm and the Go-Down-Level algorithm together can improve performance.

3.2.2 Go-Neighbor-Switch, X-Turns and Go-Down-Level Algorithms

As shown as in the **Algorithm 2**, X-Turns should be used with the Go-Neighbor-Switch algorithm. X-Turns is also used when the current switch and destination/NCA are in different groups. In short, the Go-Neighbor-Switch, X-Turns, and Go-Down-Level algorithms improve performance when they are combined.

3.2.3 Three-Turn and Go-Down-Level Algorithms

As described in Section 3.1.4, Three-Turn is also used when the current switch and destination/NCA are in different groups. That is, the Three-Turn algorithm competes with the Go-Neighbor-Switch and X-Turns algorithms. When Three-Turn is used in combination with Go-Neighbor-Switch or X-Turns, no obvious performance improvement is achieved. Therefore, performance is only improved when Three-Turn is combined with the Go-Down-Level algorithm.

3.3 Compare to DCell Local-Reroute with Proxy

DCell Fault-tolerant Routing protocol (DFR)[3] is a near-optimal, decentralized routing solution that effectively exploits the DCell structure and can effectively handle various failures. In this section, we mainly discuss the Local-route with proxy of DFR which handles in faulty links. This section compares our algorithms to the DCell Fault-tolerant Routing protocol.

Consider the following case. In a MiKANT(k, n), the source node and destination node are in the same group and there is a faulty link in the path of the source node to the NCA. For example, for a current switch $\langle 0, 1, 0, 0\rangle$ and a destination switch $\langle 0, 0, 1, *\rangle$ in MiKANT$(3, 3)$, the link $\langle 1, 1, 0, 0, 0\rangle$ is fault. It is the same as the example in Fig. 2. The current switch will choose the neighbor switch as its "proxy" and re-routes the packet to the NCA via the proxy. This is the same as the DCell Fault-tolerant Routing protocol.

Another example of using the local-reroute with proxy is for the case that the current switch reached the NCA and goes to the destination node. For example, for current switch $\langle 0, 1, 0, 0\rangle$ and a destination node $\langle 0, *, 0, 0\rangle$ in MiKANT$(3, 3)$, the link $\langle 0, 1, 0, 0, 0\rangle$ is fault. It is the same as the example in Fig. 3. The neighbor switch of the current switch will be selected as its "proxy" and re-routes the packet to the destination node via the proxy. This is also the same as the DCell Fault-tolerant Routing protocol.

That is, DCell local-reroute with proxy is a similar method to the Go-Neighbor-Switch and Go-Down-Level. But DCell local-reroute with proxy is mainly suitable for the case where the current switch can find a neighbor switch as its proxy. When all the links connected to the neighbor switches are faulty, DCell local-reroute with proxy cannot reach destination/NCA. For such a case, we further prepared other two algorithms, X-Turns and Three-Turn, to solve this problem.

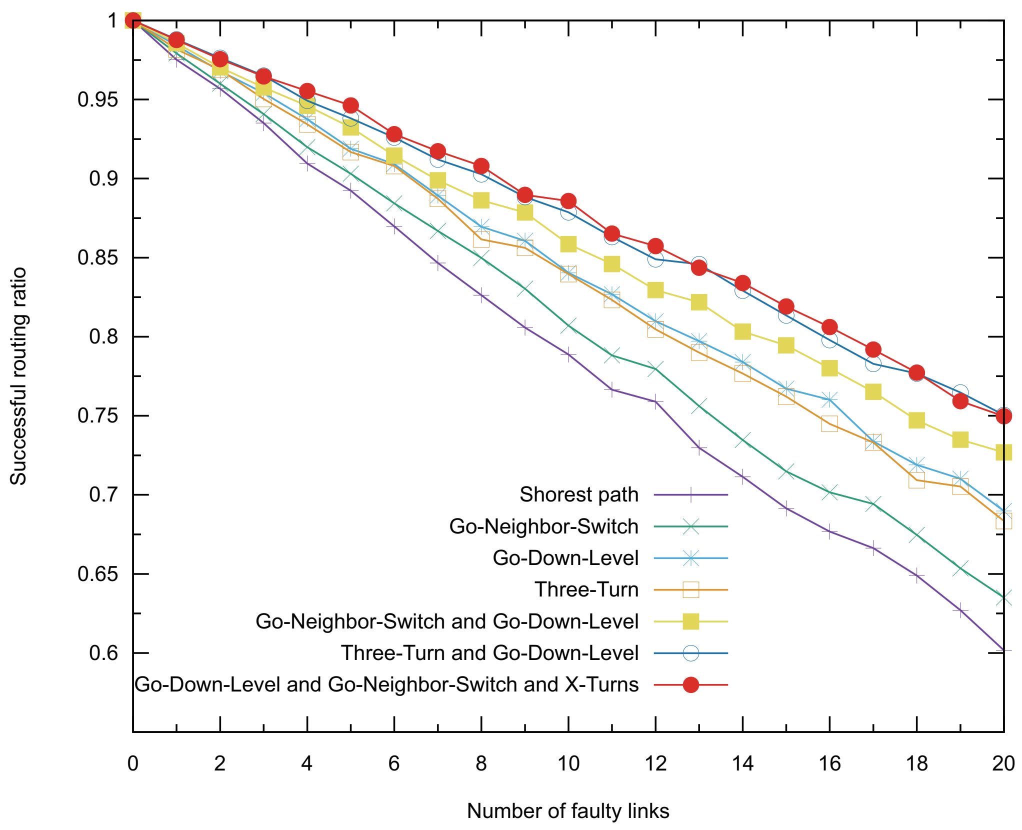

Figure 6: Successful routing ratio on link faulty

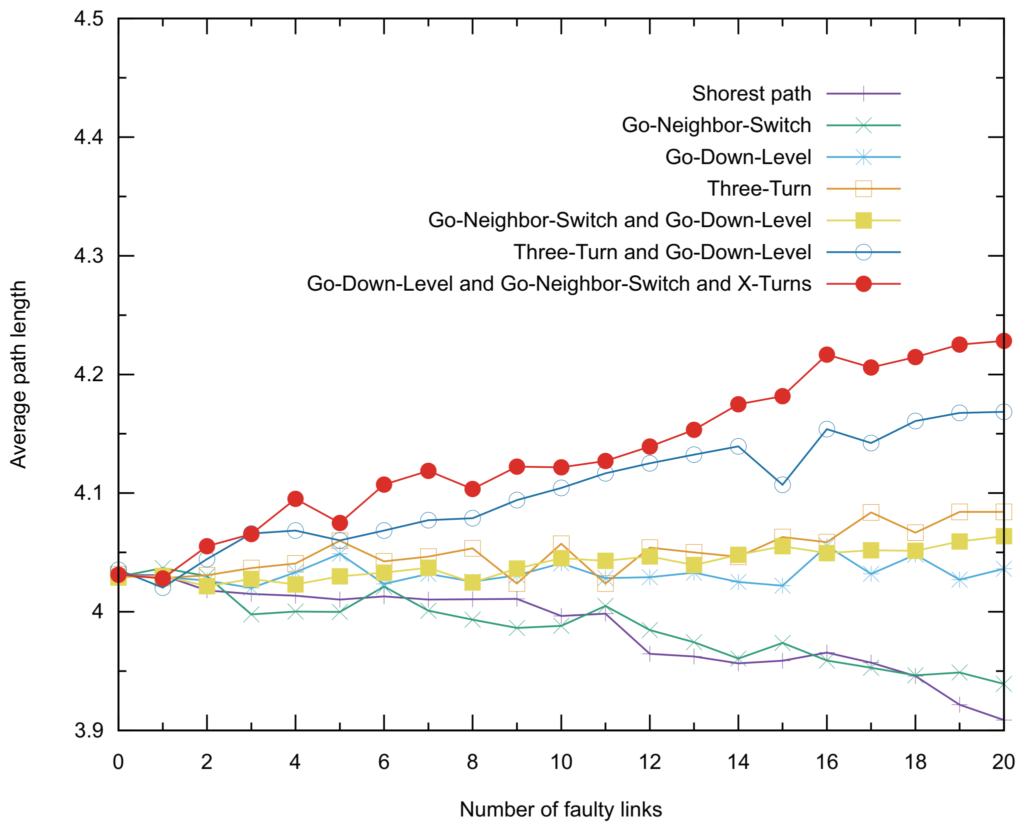

Figure 7: Average path length on link faulty

4 Evaluation of Algorithms

We have evaluated the performance of the shortest path, Go-Neighbor-Switch, Go-Down-Level, X-Turns, and Three-Turn algorithms and their combinations through simulations. We simulate the network MiKANT$(3,3)$ with 135 links. MiKANT$(3,3)$ is the smallest scale in which all the algorithms can be used and all the algorithms can be effective in a larger scale network of MiKANT(k,n). Although the mainstreams now use large-scale parallel systems, for hybrid network structures, using multiple small-scale parallel structures is also a popular method.

We developed our own simulator. For a given number of faulty links, we simulate each algorithm 100,000 times. To configure the network for each time, we randomly assign the source and destination nodes. The faulty links are also assigned randomly. One configuration is used for the simulations of all the algorithms.

Fig. 6 shows the successful routing ratios of the algorithms with the faulty links number from 0 to 20. The shortest path algorithm has the lowest successful routing ratio. The Go-Neighbor-Switch algorithm, Go-Down-Level algorithm, Three-Turn and their combination have better performance than shortest path algorithm. As shown in **Algorithm 2**, the X-Turns algorithm must be used with the Go-Neighbor-Switch algorithm. We also applied the Go-Down-Level algorithm to the X-Turns and Three-Turn simulation,respectively. From the figure, we can know the impacts of the algorithms on the successful routing ratios.

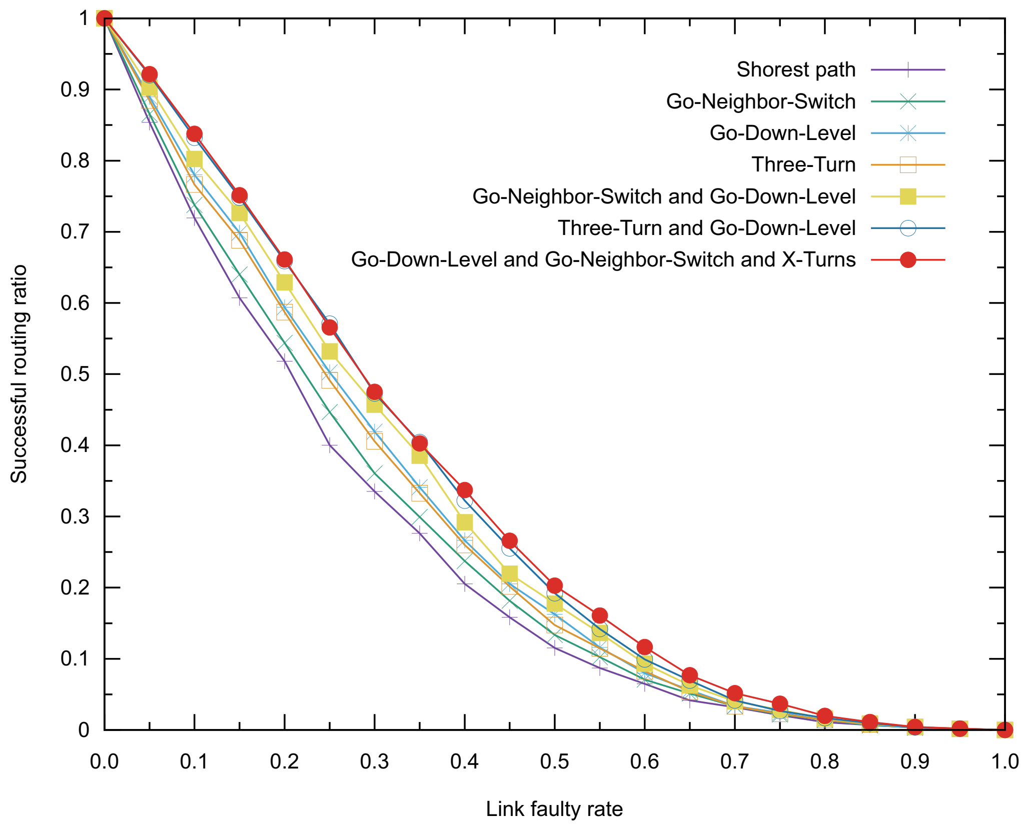

Figure 8: Successful routing ratio on link faulty

Fig. 7 shows the average path length when the routings are successful. 20 faulty links in MiKANT$(3,3)$ are less than 15% of total links, the improvement of successful routing ratio in Go-Neighbor-Switch is not obvious. Because we only count the path length when the routing is successful, and some routings with shorter paths may be successful by using the Go-Neighbor-Switch algorithm but such routings are failed by using the shortest path algorithm, the average path length of Go-Neighbor-Switch may be smaller than that of shortest path. Go-Down-Level algorithm, Three-Turn algorithm and X-Turns algorithm increase path lengths obviously. Go-Down-Level algorithm and Three-Turn algorithm have almost the same path length. Generally, the algorithm achieves higher successful routing ratio has a longer routing path.

Fig. 8 shows the successful routing ratios of the algorithms with the link faulty rate ranging from 0% to 100%. The combined algorithms show better performance. Faulty rate of 100% means that all the links are faulty. This does not happen for a well-maintained system.

Fig. 9 shows the average path length when the routings are successful with the link faulty rate ranging from 0% to 100%. We can find that the path length increases and then decreases as the link faulty rate increases. This is because, when the link faulty rate is high, the routing can be successful only when the source node and destination node have a shorter distance between each other.

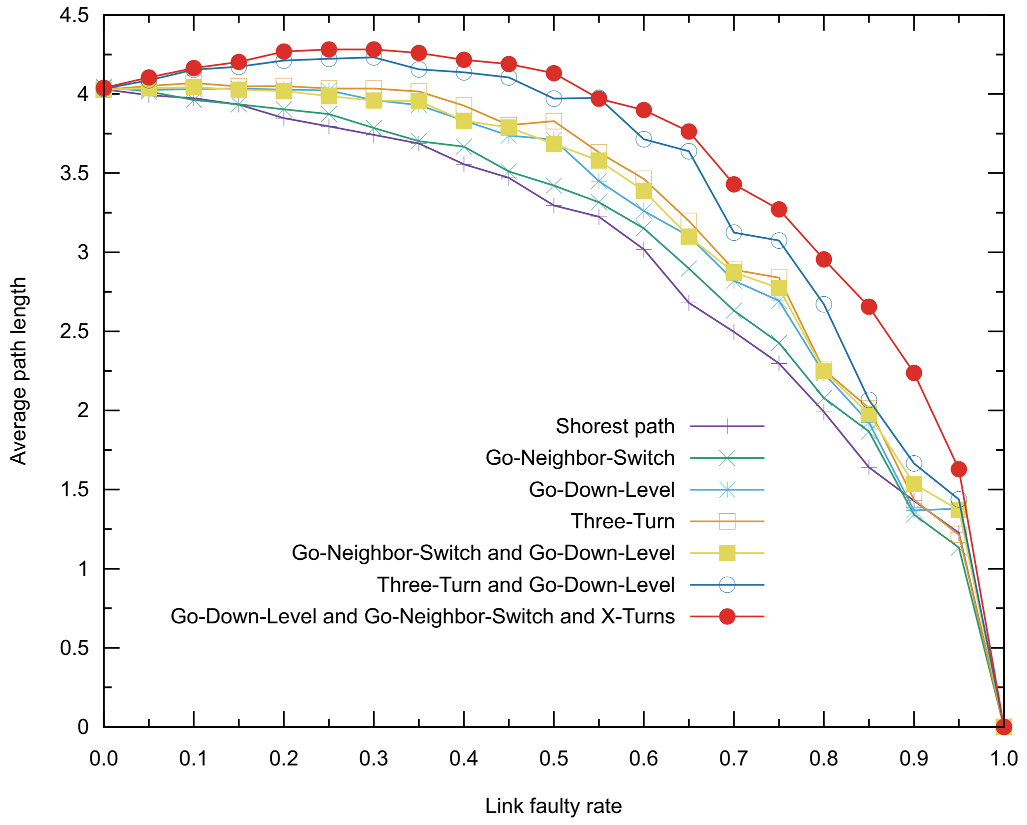

Figure 9: Average path length on link faulty

5 Conclusions

As one of the variants of fat-trees, MiKANT network can achieve higher performance at lower implementation cost. As the scale of MiKANT becomes large, the probability of link failure increases. Fault tolerance is one of the important issues of the interconnection networks for large-scale parallel computers. The four algorithms and their combinations proposed in this paper can find a path from the source node to the destination node at high probability in the MiKANT with faulty links. The future research work may include developing new link fault tolerant algorithms of higher performance and evaluating the performance of the algorithms for both the switch and link fault tolerance in MiKANT.

References

[1] M. Adda and A. Peratikou. Routing and fault tolerance in z-fat tree. *IEEE Transactions on Parallel and Distributed Systems*, 28(8):2373–2386, 2017.

[2] J. Gliksberg, A. Capra, A. Louvet, P. J. García, and D. Sohier. High-quality fault-resiliency in fat-tree networks (extended abstract). In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 9–12, 2019.

[3] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: A scalable and fault-tolerant network structure for data centers. In *SIGCOMM08*. Association for Computing Machinery, Inc., August 2008.

[4] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[5] 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 *The 18th IEEE International Conference on Scalable Computing and Communications*, pages 1643–1650, Oct. 2018.

[6] Y. Li and W. Chu. Switch fault tolerance in a mirrored k-ary n-tree. In *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5, 2019.

[7] Javier Navaridas, Jose Miguel-Alonso, Francisco Javier Ridruejo, and Wolfgang Denzel. Reducing complexity in tree-like computer interconnection networks. *Parallel Comput.*, 36(2–3):71–85, February 2010.

[8] Nitin and Durg Singh Chauhan. Comparative analysis of traffic patterns on k-ary n-tree using adaptive algorithms based on burton normal form. *J. Supercomput.*, 59(2):569–588, February 2012.

[9] F. Petrini and M. Vanneschi. k-ary n-trees: high performance networks for massively parallel architectures. In *Proceedings 11th International Parallel Processing Symposium*, pages 87–93, Apr. 1997.

[10] 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*, pages 153–160, 2008.

[11] F. O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato. Dynamic fault tolerance in fat trees. *IEEE Transactions on Computers*, 60(4):508–525, Apr. 2011.

[12] W. Xiaolin, Q. Xiaogang, and L. Lifang. Probe selection algorithm for faulty links localization in all-optical networks. In *2017 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, pages 269–272, 2017.