Journal of Marine Science and Technology
DOI: 10.6119/JMST-013-0522-2

A FAULT-TOLERANT OPTIMAL MESSAGE ROUTING METHODOLOGY FOR CUBE-CONNECTED-CYCLES PARALLEL COMPUTERS

Gene Eu Jan[2], Cheng-Hung Li[1], Yung-Yuan Chen[2], and Shao-Wei Leu[1]

Key words: cube-connected-cycles, parallel computer, interconnection network, backtracking, radiation.

ABSTRACT

A cube-connected-cycles (CCC) is a regular graph, suitable for constructing the interconnection network of parallel or multi-node computer systems. The CCC network possesses the features of symmetry, regularity, fault tolerance, and fixed degree of the network. Message delivery through an interconnection network sometimes fails due to processing node and/or link faults or simply because some processors are too busy to handle message transfer. To make a CCC-based parallel computer more resilient to node/link faults, this paper proposes an optimal routing method which guarantees to find a shortest path between the source and the destination nodes within a faulty CCC-based network if such a path exists. The proposed method is based on the concepts of radiation and backtracking and is able to find the shortest path with little impact on the network traffic load. The time complexity of our routing methodology is $O(\log N)$, where N is the number of nodes in the CCC architecture.

I. INTRODUCTION

The cube-connected-cycles (CCC) was introduced by Preparate and Vuillemin [11] as an efficient network topology suitable for general-purpose parallel computing. It has a number of useful properties, such as symmetry, regularity, $O(\log N)$ diameter and robustness. The CCC network is an extension of the hypercube by replacing each node with a cycle of n nodes, provided that the hypercube is n-dimensional. The network diameter of an n-CCC equals $2n$. The major improvement of a CCC is that its degree is fixed at three, independent of the dimension of the underlying hypercube. Given the same number of nodes, a CCC network has smaller diameter and fewer links, hence lower cost (defined as degree × diameter) than mesh or hypercube topologies.

For the CCC network to be useful in constructing high-performance parallel computers, it requires some efficient mechanism for message routing, so that processing nodes participating in parallel computation can exchange messages as quickly as possible. Moreover, as the size of the CCC-based parallel system gets larger, the chance that some messages may fail to get to their intended destinations due to node/link failures also increases. Therefore, it is of great importance that the design of a routing mechanism takes into account the presence of node/link faults.

Routing is a type of mechanism that is of critical importance to the successful operations of parallel computers. The objective of a routing mechanism is to ensure that packets/messages can be delivered successfully from the source node to one or more destination nodes within prescribed time limits. In the presence of node or link faults, packets/messages may not be able to reach the destination unless the routing mechanism is capable of finding one or more alternative routes. In what follows we review some CCC-related routing mechanisms from the literature [2, 5, 12, 14, 15].

In general, routing mechanisms for the CCC can be classified into three categories: unicast, multicast, and broadcast. For unicast routing, Meliksetian and Chen [9] first constructed a special CCC(d, d) network, where d is the dimension of the underlying hypercube, and obtained an optimal routing algorithm for the network. The authors then generalized the said method for CCC(d, k) networks, where k is the number of nodes per cycle, and obtained a path length of $2d + d/2 - 2$, which happened to be the diameter of the network.

Beyond unicast CCC routing, Song et al. [13] suggested an optimal multicast algorithm for n-dimensional CCCs based on the property of switching technology and virtual channels. Their algorithm can multicast a message to $m-1$ targets in $\log_2 m$ steps. Jang [7] proposed a binomial tree based optimal broadcasting algorithm for CCCs, which took $1+2(\lceil h-1/2 \rceil)+ s(1+\lceil h-1/2 \rceil)$ steps to broadcast a message to all other nodes with the ability to tolerate $s-1$ node or link failures, where s is the dimension of a CCC and h is the number of nodes in each cycle. Fang et al. [3] developed two closely related broadcasting algorithms based on the binomial

Paper submitted 10/08/12; revised 04/26/13; accepted 05/22/13. Author for correspondence: Shao-Wei Leu (e-mail: b0119@mail.ntou.edu.tw).
[1]*Department of Electrical Engineering, National Taiwan Ocean University, Keelung, Taiwan.*
[2]*Graduate Institute of Electrical Engineering, National Taipei University, New Taipei City, Taiwan.*

tree for *n*-dimensional composite CCC and bi-directional CCC. Their methods take $3n-2$ steps to broadcast a message to all non-faulty nodes for the composite CCC and $2n-1+\lceil (n-1)/2 \rceil$ steps for the bi-directional variant.

Inspired by Gaughan and Yalamanchili's treatment of hypercube routing [4], our unicast CCC routing method also involves two phases: a radiation phase to search for the destination node and a backtracking phase to establish a path leading back to the source node. In both phases, only a small routing token consisting of mainly a few addresses is being transmitted from node to node, incurring little communication load on the network. When a node receives a routing token, it acts according to the content of the token. If it's necessary for the token to continue its journey, proper modification of the token must be done first. The successful completion of the two-phase operation establishes an optimal route for message transfer between the source and the destination nodes in the presence of node or link faults.

The rest of the paper is organized as follows. In Section 2, we introduce the architecture of the CCC. Section 3 presents the proposed fault-tolerant routing algorithm. Section 4 provides detailed graphical illustrations of the algorithm. A performance analysis is given in Section 5, and Section 6 concludes the paper.

II. THE CCC ARCHITECTURE

Improving on the basis of the hypercube, the CCC topology is a hybrid of the hypercube and the ring graph. A CCC of *n* dimensions has $N = n \times 2^n$ nodes, $E = 3n \times 2^{n-1} = 3N/2$ connecting links, and a diameter of $2n-1+\lfloor n/2 \rfloor$ for $n = 3$ and $2n+\lfloor n/2 \rfloor - 2$ for $n \geq 4$. It is obvious that $E = O(N)$ and $D = O(\log N)$ for the CCC. As Fig. 1 shows, disregarding the address tags in illustration for now, each and every node of a 3-dimensional hypercube is replaced by a ring of three nodes. The resulting structure is a 3-dimensional CCC graph, where *N* = 24, *E* = 36, and *D* = 6 [1, 6, 8, 10].

The CCC can efficiently solve a large class of problems including the Fourier transform, merging, sorting, permutations, and several matrix operations. The CCC not only preserves all the attractive features of the hypercube, such as small diameters, large bisection widths, and symmetry, but also has fewer links and smaller constant node degrees. For the same number of dimensions, a CCC has *n* times as many nodes as a hypercube does. Moreover, assuming equal number of nodes, a CCC has lower dimensions, fewer connecting links, and a larger network diameter. For example, a CCC of four dimensions has 64 nodes, while a hypercube needs six dimensions to include the same number of nodes. A four-dimensional CCC has 96 connecting links, which is much fewer than the 384 connecting links in a six-dimensional hypercube constructed out of the same number of nodes. However, the network diameter of a four-dimensional CCC is 8, which is greater than that of a six-dimensional hypercube by a margin of 2.

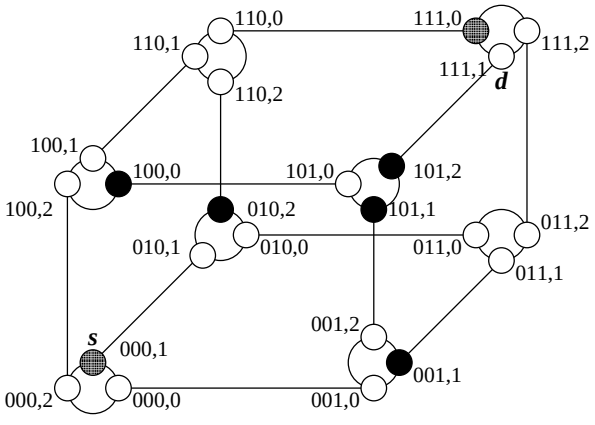


Fig. 1. Structure of a CCC graph.

An *n*-dimensional CCC consisting of $(n \times 2^n)$ nodes can be denoted by $\mathrm{CCC}(n, 2^n)$. Each node can be indexed by a pair of numbers (x, y), where $0 \leq x \leq 2^n - 1$ and $0 \leq y \leq n-1$. A node (x, y) is connected to three neighboring nodes denoted as $(x, (y+1) \bmod n)$, $(x, (y-1) \bmod n)$, and $(x \oplus 2^y, y)$, where $\oplus$ is the bitwise XOR operation. As an illustration for the notation defined above, Fig. 1 shows the structure of a $\mathrm{CCC}(3, 2^3)$ network with each node being labeled with its index or address. Note that, for all illustrations in this paper, the first number of the index (x) is in binary form to reflect the topological extension from the hypercube.

III. FAULT-TOLERANT ROUTING ALGORITHM

For a multicomputer/multiprocessor system based on the CCC topology, information may be exchanged frequently among the computing nodes through the interconnecting links across the whole architecture. In order to deliver the messages from the source node to the destination node as quickly as possible, a message routing algorithm must be able to find a shortest path, which bypasses all broken links and faulty/busy processing nodes.

As shown in Fig. 2, if all the nodes and links are healthy, there exist four shortest paths, each taking six steps to go from the source node to the destination node. However, in the presence of two faulty nodes and one broken link, only one shortest path remains, which is depicted with thick directed edges. Achieving reliable message transfer in the presence of faults is imperative to the successful deployment of any interconnection network. In this study, we propose a fault-tolerant routing methodology for cube-connected-cycles to determine the shortest path between the source and destination nodes, based on the concepts of radiation and backtracking.

The concept of radiation is described as follows. The process of radiation begins at the source node *s* where a routing token is duplicated and sent to each adjacent neighboring node. The routing token consists of the routing phase, the addresses of both the source and destination nodes, the address of the

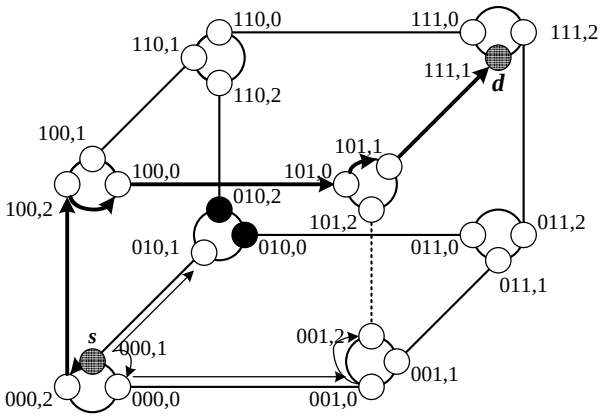


Fig. 2. A CCC network with two faulty nodes in dark color and one broken link depicted as a dashed line.

current address, **(*ca*)**	distance, **(*dist*)**
preceding address, **(*pa*)**	next address, **(*na*)**

Fig. 3. Contents of the routing table.

sending node, and the distance between the sending node and the source node. The adjacent nodes that receive the tokens continue the radiation process until the destination node is reached.

Each node uses a routing table to save the routing information, which can be used to create the shortest path between the source and destination pair. The routing table contains four fields as shown in Fig. 3. The current address field *ca* stores the address of current node. The distance field *dist* stores the distance between the current node and the source node. The preceding address field *pa* denotes the address of the preceding node, from which the current node received the token. During the radiation phase, the *pa* field is set only by the first incoming token. Once it has been set, other incoming tokens going to the destination will be discarded to prevent further transmission of useless tokens. The *pa* information will be used in the backtracking process to find the next node one step closer to the source. Furthermore, during the radiation phase, the *dist* field is updated by adding one to the distance value contained in the received token.

During the backtracking phase, each node uses the address of the sending node in the token to update the *na* field of its routing table, which indicates the next node to send the message during the message transmission phase. The radiation process can tolerate the existence of faulty nodes and broken links and guarantees to find the shortest path between the source and destination nodes if at least one exists. If no such path exists between the source and the destination due to faults that occur in the CCC, the token will not reach the destination during the radiation phase. It is also possible that a backtracking token is blocked by a broken link before reaching the source. In both cases, the routing request will be terminated after the routing time exceeds an upper bound, which is in general two times the diameter of the CCC.

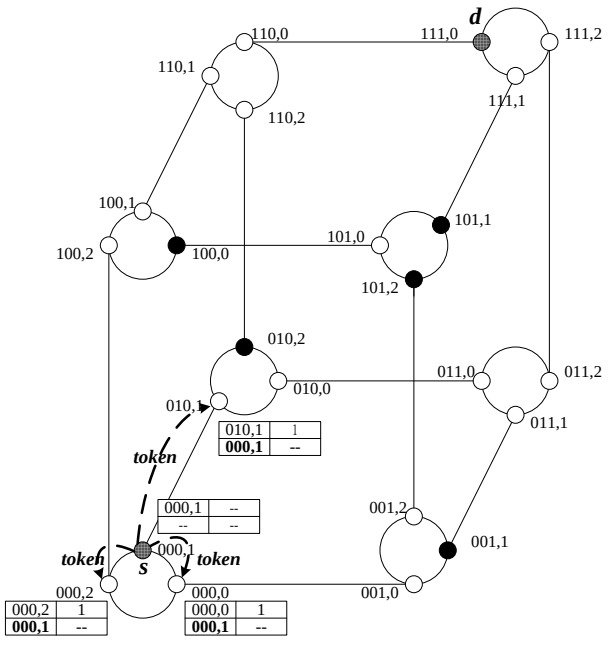
Table 1. Notations and Variables.

Notation	Definition
sa	address of the source node
da	address of the destination node
token={***rp***, ***send_a***, ***source_a***, ***dest_a***, ***dss***}	routing information sent by a node to its adjacent nodes, consisting of: ***rp***: routing phase, possibly in radiation phase and backtracking phase ***send_a***: address of the sending node ***source_a***: address of the source node ***dest_a***: address of the destination node ***dss***: distance between the sending node and the source node
routing_ table ={***ca***, ***dist***, ***pa***, ***na***}	***ca***: address of current node ***dist***: distance between current node and source node ***pa***: address of the preceding node ***na***: address of the subsequent node

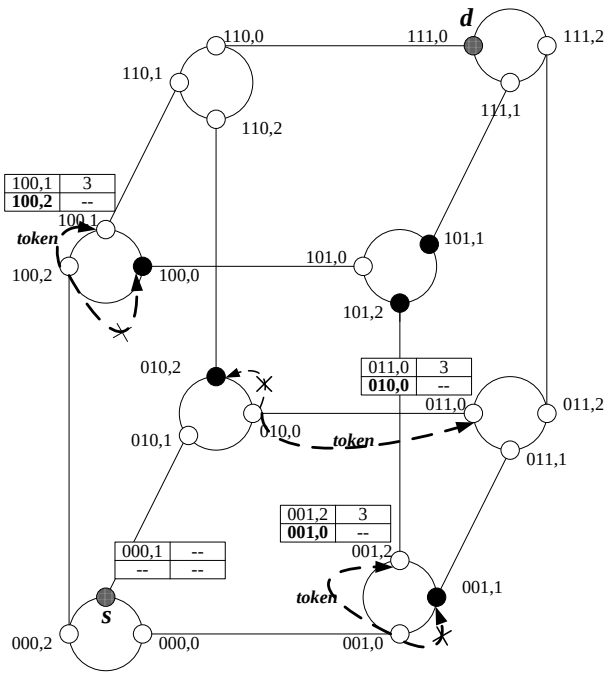
In summary, a routing request includes three phases: radiation, backtracking, and message transmission phases. The radiation phase is responsible for finding the shortest path between the source and destination nodes, as well as creating a routing table for each node along the path. The radiation phase continues until the destination node is reached. During the radiation phase, each node can save the address of its preceding node in the routing table. This information is later used in the backtracking phase to indicate the next node leading back to the source. The backtracking phase is activated as soon as the destination node is reached. During the backtracking phase, each node on the shortest path saves the address of its previous node in the *na* field of its routing table. That information later becomes the pointer to the next node during the message transmission phase. Table 1 defines all variables used in this paper. The fault-tolerant routing algorithm concurrently executed in all CCC nodes is described as follows.

CCC-Single-Packet-Optimal-Routing

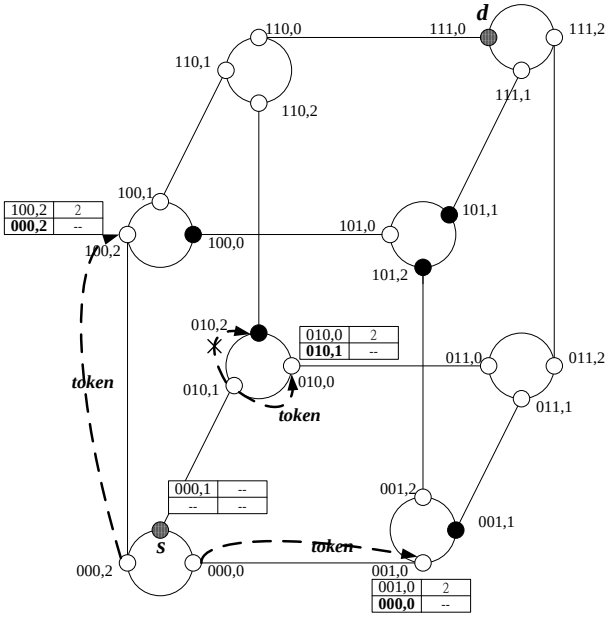
```
1. if the node receives no token then
2.    if the node wants to initiate routing then
3.       PREPARE_TOKEN( )
4.       Send token to all neighbors.
5. else
6.    if node is destination then
7.       if token is in radiation phase then
8.          Update routing table.
9.          Send token to the node indicated by pa field of routing table.
10.      else /* token is in backtracking phase */
11.         Store send_a field of token in na field of routing table.
12.         Start message transfer operation.
13.   else    /* node is not destination */
14.      if token is in radiation phase then
15.         if pa field of routing table not yet assigned in current cycle then
16.            RADIATION_UPDATE( )
17.            Send token to all neighbors.
```



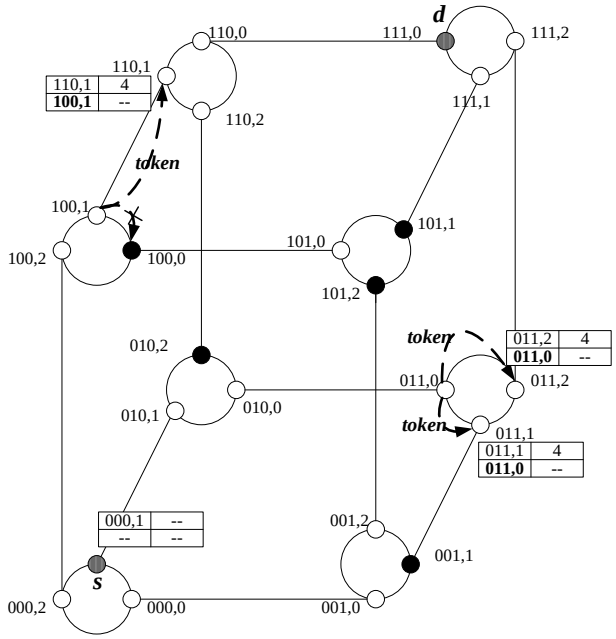
(a)



(c)



(b)



(d)

18. **else** /* *pa* field of routing table has already been assigned */
19. Discard token.
20. **else** /* token is in backtracking phase */
21. Store *ca* field of routing table in *send_a* field of token.
22. Store *send_a* field of token in *na* field of routing table.
23. Send token to the node indicated by *na* field of routing table.

PREPARE_TOKEN( )

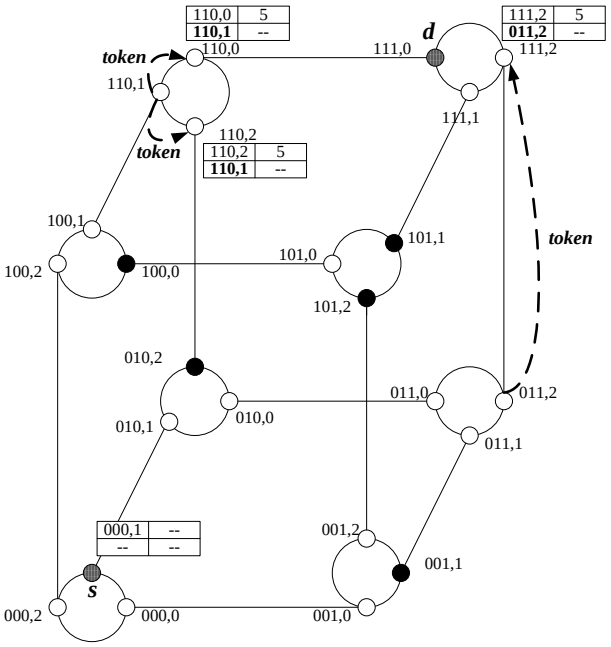
1. *token.send_a* ← *routing_table.ca*;
2. *token.source_a* ← *sa*;
3. *token.dest_a* ← *da*;
4. *token.dss* ← 0;

RADIATION_UPDATE( )

1. *routing_table.pa* ← *token.send_a*;
2. *routing_table.dist* ← *token.dss* + 1;
3. *token.send_a* ← *routing_table.ca*;
4. *token.dss* ← *routing_table.dist*;

IV. ILLUSTRATION OF THE PROPOSED ROUTING ALGORITHM

To explain the proposed routing algorithm in detail, Fig. 4 through 6 serve to illustrate the operations of the radiation



(e)

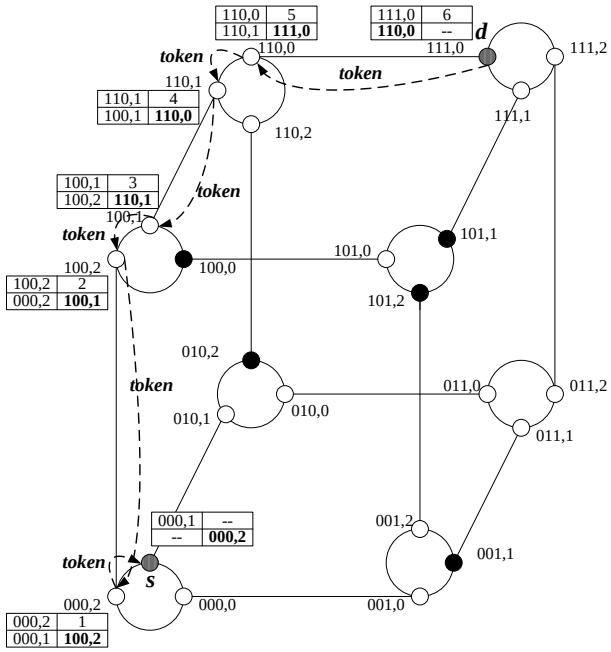
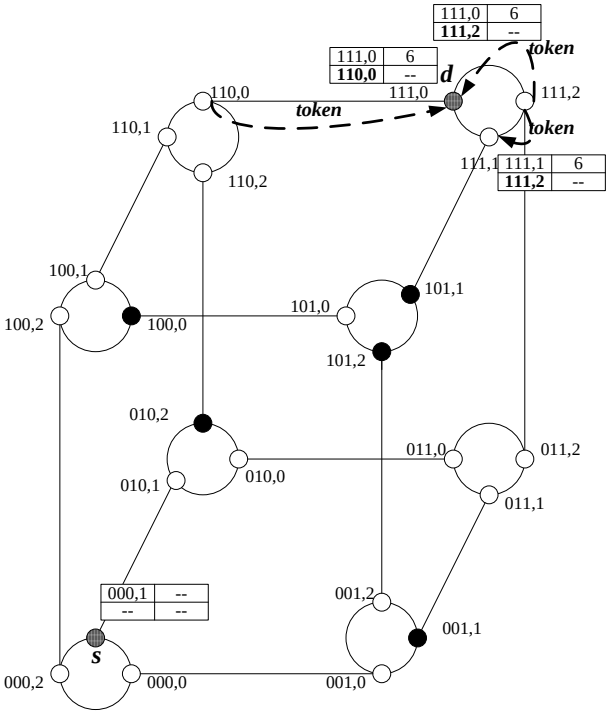


Fig. 5. The corresponding backtracking phase.



(f)

Fig. 4. Illustrations of an example radiation phase.

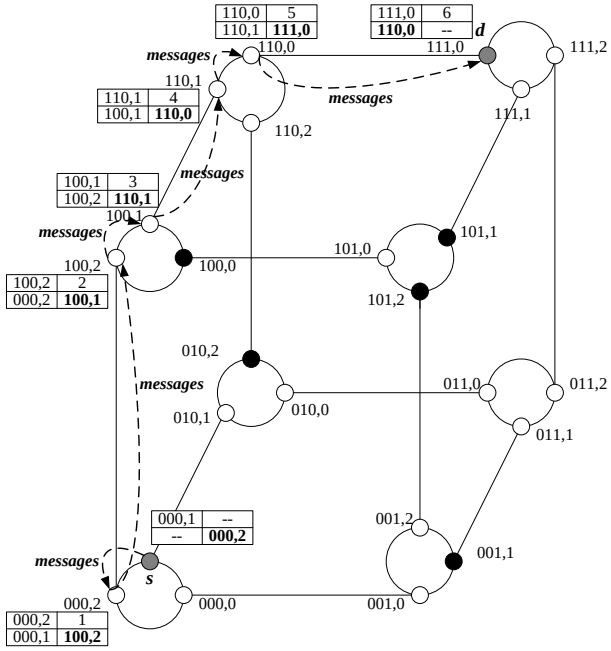


Fig. 6. The resulting message transmission phase.

phase, the backtracking phase, and the message transmission phase, respectively. Also, in every illustration, the nodes labeled with *s* and *d* are the source and the destination nodes, respectively. Fig. 4 explains the operation of the radiation phase in six steps. In Fig. 4(a), the source node *s* duplicates and radiates tokens to its adjacent nodes. Each adjacent node receives the token and stores the address of the sending node (*send_a*) in the *pa* field of the routing table. The distance between the current node and the source node is 1. In Fig. 4(b), each node which received a token in the last step, i.e. nodes (000, 0), (000, 2), or (010, 1), now radiates the token to its adjacent nodes, and each successful delivery of the token triggers update of the routing table at the receiving node, including nodes (001, 0), (010, 0), and (100, 2). Note that a faulty or busy node can neither receive nor radiate any token. The radiation process continues as illustrated in Fig. 4(c) through 4(f) until the destination is reached. When this hap-

pens, the distance field becomes 6 at the destination node.

The destination node activates the backtracking phase after it has received a radiation token targeted to itself. Backtracking from destination node *d* to source node *s* can be achieved through using the *pa* field of the routing table. The process is described as follows. The destination node sends the backtracking token to its preceding node, as indicated by the *pa* field of its routing table. The receiver of the backtracking token stores the address of the sending node in the *na* field of its routing table. The backtracking token then gets updated and sent to the preceding node again. This process repeats until the source node is reached. If there exist more than one shortest paths, then any one of them may be selected as the backtracking route, as shown in Fig. 5.

The message transmission phase starts when the source node receives the backtracking token. From the source node the message is sent to the node pointed to by the *na* field of the routing table. The message-sending operation is repeated at every node on the established path until the message is delivered to the destination node. As shown in Fig. 6, our routing methodology is able to deliver the whole message from the source node *s* to the destination node *d* in the presence of faulty nodes and broken links.

V. PERFORMANCE ANALYSIS

This section evaluates performance of the proposed routing algorithm in terms of the number of steps required to establish a shortest path. When $n = 3$, the radiation phase starting from the source needs a maximum of $2n-1+\lfloor n/2 \rfloor$ steps to reach the destination. Therefore, the time complexity of the steps required is $O(\log N)$, where N is the number of nodes. When $n \geq 4$, it takes a maximum of $2n+\lfloor n/2 \rfloor - 2$ steps to reach the destination, hence the time complexity stays as $O(\log N)$. Since the operation of the backtracking phase is similar to the radiation phase, being different only in the direction, the time complexity derived for the radiation phase is entirely valid for the backtracking phase. As a result, the time complexity for the operation to establish a shortest path remains as $O(\log N)$.

VI. CONCLUSION

This paper proposes an effective fault-tolerant routing methodology for the CCC network, which guarantees to establish a shortest path between the source node and the destination node if at least one such path exists. Most other proposed message routing algorithms for the CCC network send the entire message in every step to establish an optimal route, thus causing extra traffic load in many parts within the network. In contract, the token adopted in our method contains only simple information and has little impact on the traffic and the time complexity is kept at $O(\log N)$. In the future, our research will base on the concept of radiation to establish shortest paths from one source node to multiple destination nodes within a faulty CCC.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers whose valuable comments greatly helped to improve the content of the paper.

REFERENCES

1. Dandamudi, S. P., *Hierarchical Hypercube Multicomputer Interconnection Networks*, Ellis Horwood (1991).
2. Dauto, J., Yalamanchili, S., and Ni, L. M., *Interconnection Networks: An Engineering Approach, Morgan Kaufmann*, San Francisco (2003).
3. Fang, J.-F., Lee, C.-M., Yen, E.-Y., Chen, R.-X., and Feng, Y.-C., "Novel broadcasting schemes on cube-connected cycles," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp.629-632 (2005).
4. Gaughan, P. T. and Yalamanchili, S., "Adaptive routing protocols for hypercube interconnection networks," *IEEE Computer*, Vol. 26, No. 5, pp. 12-23 (1993).
5. Hsu, L. H. and Lin, C. K., *Graph Theory and Interconnection Networks*, CRC Press (2009).
6. Hwang, K., *Advanced Computer Architecture; Parallelism, Scalability, Programmability*, McGraw-Hill (1993).
7. Jang, J. E., "An optimal fault-tolerant broadcasting algorithm for a cube-connected cycles multiprocessor," *International Conference on Databases, Parallel Architectures and Their Application*, pp. 206-215 (1990).
8. Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Francisco (1992).
9. Meliksetian, D. S., and Chen, C. Y. R., "Optimal routing algorithm and the diameter of the cube-connected cycles," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 10, pp. 1172-1178 (1993).
10. Parhami, B., *Introduction to Parallel Processing: Algorithms and Architectures* (1999).
11. Preparata, F. P. and Vuillemin, J., "The cube-connected cycles: a versatile network for parallel computation," *Communications of the ACM*, Vol. 24, No. 5, 300-309 (1981).
12. Scherson, I. D. and Youssef, A. S., *Interconnection Networks for High-Performance Parallel Computers*, IEEE Computer Society Press (1994).
13. Song, J., Hou, Z., and Shi, Y., "An optimal multicast algorithm for cube-connected cycles," *Journal of Computer Science and Technology*, Vol. 15, Issue 6, pp. 572-583 (2000).
14. Wu, C.-L. and Feng, T.-Y., *Tutorial: Interconnection Networks for Parallel and Distributed Processing*, IEEE Computer Society Press (1984).
15. Xu, J., *Topological Structure and Analysis of Interconnection Networks*, Kluwer Academic Publishers (2001).
16. Zomaya, A. Y., *Parallel and Distributed Computing Handbook*, McGraw-Hill (1996).