FTRH: Fault Tolerance Routing Algorithm for Hex-Cell Networks

Mohammad Qatawneh[1], Bdour Hamed[2], Wesam AlMobaideen[1], Azzam Sleit[1], Amal Oudat[1], Wala'a Qutechat[1], Roba Al-Soub[1]

[1]Computer Science Department, King Abdulla II School for Information Technology, P.O. Box 13898
University of Jordan, Amman 11942, Jordan
[2] *Computer Science Department, Mutah University, Karak, Jordan*

Summary
This paper describes a new fault tolerance routing algorithm for Hex-Cell networks. Hex-Cell is an interconnection network topology that employs an efficient routing algorithm and combines attractive features which make it a good candidate to be used for many applications. The Fault Tolerance Routing algorithm for Hex-cell networks (FTRH) presented in this paper focuses on component software failures and guaranteed message delivery from source to destination even with the presence node and link failure. Through the analysis of the algorithm, we demonstrate that the proposed routing protocol finds routing paths that are close to optimal in most cases.
Key words:
Hex-Cell, Fault-Tolerance Routing, Network Topology, Parallel Processing.

1. Introduction

Interest in massive parallel processing has increased rapidly boosting the need for larger number of interconnected processors. It has been shown that as the number of interconnected processors rises, the probability of having faulty nodes increases and it becomes essential to find communication paths which detour faulty processors or links [2, 3, 4, 15]. In distributed and parallel processing systems with faulty processors, it is very important to select shortest paths to support efficient interprocess communication.
If every processor in the system identifies the status of all processors, an optimal routing is possible which is very hard to adopt due to restrictions of space and time complexities. Fault-tolerance is the property that enables a system to continue operating properly in the event of failure of some of its components such as a node fault, link fault or both [2, 8, 13]. Each component may suffer from hardware failure or software failure. The system which doesn't deal with faulty problem may be unreliable, inefficient and can suffer from higher latency [8, 14].
We focus on a dynamic routing scheme of interconnected processors, which communicates messages in a faulty hex-cell in order to suppress performance degradation [1, 4]. In this paper, we introduce a new fault tolerance routing algorithm for hex-cell networks which deals with the software component failure problems. A hex-cell network combines desired topological features like less communication cost, efficient routing, and the capability of embedding static topologies such as linear array, ring, tree, and mesh topology [1, 3]. Adding fault tolerance features to these attractive features increases reliability, availability, and efficiency. Some fault tolerance routing algorithms wait for failure to happen then react accordingly [7, 8, 10]. The proposed Fault Tolerance Routing protocol for Hex-Cell networks (FTRH) monitors the network status to choose the best path before taking the decision of routing the message. The aim is to avoid message rerouting in order to reduce the imposed routing delay.
The rest of this paper is organized as follows. Section 2 discusses the fault tolerance problem. Section 3 presents the definition of Hex-Cell and its routing algorithm. Section 4 proposes the fault tolerance routing algorithm. Section 5 shows the results and discussion. Finally in Section 6, some concluding remarks are made.

2. Related Work

Many researches have addressed the fault tolerance problem for various network topologies. In [5], a new routing methodology for tori and meshes topologies was proposed to achieve high performance without the use of virtual channels. The Segment-based Routing (SR) algorithm handles any topology derived from any combination of faults when combined with static configuration. This algorithm partitions a topology into subnets and subnets into segments which places bidirectional turn restrictions locally inside a segment. The introduction of a locality independence property results with a larger degree of freedom in the placement of routing restrictions when compared with other routing strategies. Evaluation results have shown performance superiority of SR especially in the presence of link failures.
In [6], a routing scheme was presented to prove that fault-tolerance in hypercube topology networks can be

Manuscript received December 5, 2009
Manuscript revised December 20, 2009

achieved. Comparisons of the proposed routing scheme, namely Mask Interval Routing Scheme (MIRS) with other classical routing protocols have proved flexibility. In [9], the authors defined a 2D hexagonal mesh multiple interconnection networks based on triangular tessellation, and presented a 3d hexagonal mesh. Although the 2D hexagonal mesh topological properties are well known, the existing addressing schemes can not be extended to a 3D hexagonal mesh. They presented an addressing scheme and an optimal routing algorithm for a 2D hexagonal network. Additionally, a new 3D hexagonal network that can be extended as a natural generalization from the 2D hexagonal mesh was proposed.

3. The Hex-Cell Network Topology

A Hex-Cell network [1] with depth d is denoted by HC(d) and can be constructed by using units of hexagon cells each consisting of six nodes. A Hex-Cell network with depth d has d levels numbered from 1 to d, where, level 1 represents the innermost level corresponding to one hexagon cell. Level 2 corresponds to the six hexagon cells surrounding the hexagon at level 1. Level 3 corresponds to the 12 hexagon cells surrounding the six hexagons at level 2 as shown in Fig. 1. The levels of the HC(d) network are labeled from 1 to d. Each level **i** has $\mathbf{N_i}$ nodes, where $\mathbf{N_i} = 6(2i-1)$.

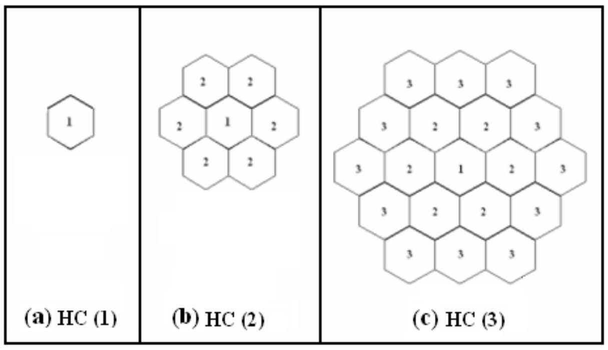


Fig. 1 (a) HC (one level) (b) HC (two levels) (c) HC (three levels)

In this section we describe the routing algorithm for hex-cell network which introduced in [1]. Each node in the HC is identified by a pair **(X, Y)**, where **X** denotes the line number in which the node exists, and **Y** denotes the location of the node in the line as shown in Fig. 2. A node with the address (1, 1) is the first node that exists at line number 1. (1, 2) refers to the second node that exists at line number 1, and so on.

Assume that X_s is the line number of the source node, Y_s is the location of source node in line, X_d is the line number of destination node, and Y_d is the location of destination node in the line. One of the following cases will be called recursively until the destination has been reached.

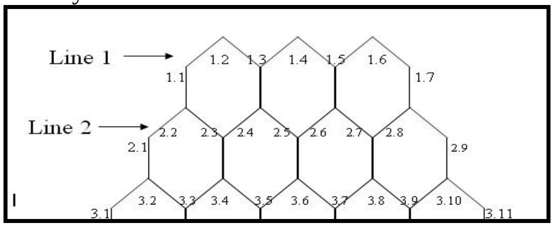


Fig. 2 Addressing Node in Hex-Cell

Case 1: $(X_s > X_d) \rightarrow$ moveUp(X_s,Y_s,X_d,Y_d): we have two directions; namely, moveUp/Left-to-Right, and moveUp/Right-to-Left.
Case 2: $(X_s < X_d) \rightarrow$ moveDown(X_s,Y_s, X_d,Y_d): we have two directions; namely, moveDown/Left-to-Right and moveDown/Right-to-Left.
Case 3: $(X_s = X_d) \rightarrow$ moveHorizontal(X_s,Y_s, X_d,Y_d): we have two directions; namely, moveHorizontal/Left-to-Right and moveHorizontal/Right-to-Left.

Fig. 3 shows how the routing algorithm for Hex-Cell works for a non-faulty network. Let (X_s,Ys) = (4, 9) be the source node and (X_d,Y_d) = (2, 4) be the destination node. When executing the routing algorithm, case 1 will be applied (MoveUp) {(3, 9)→(3, 10)→(2, 9)}, then case 3 (MoveHorizontal/Right-to-Left) will be applied {(2, 8)→(2, 7)→(2, 6)→ (2, 5)→(2, 4)}.

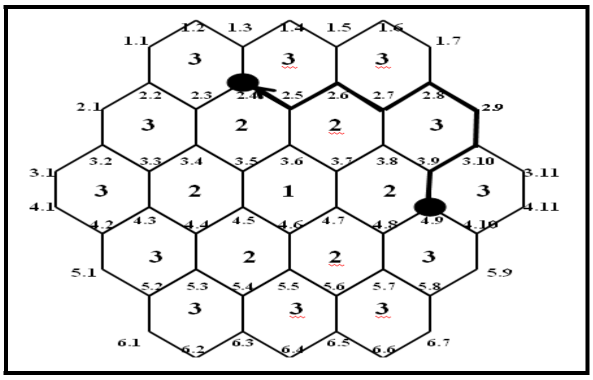


Fig. 3 Routing in a non-faulty network.

4. Fault Tolerance Routing (FTRH)

This section presents the Fault Tolerance Routing Algorithm (FTRH) for the Hex-Cell topology in details. The faulty model is discussed followed by an explanation for the routing algorithm.

4.1 FTRH Model

The algorithm considers node and link failures such that all links incident to a faulty node are considered faulty. Status signals are sent on the physical channel continuously and are monitored by a designated processor. All the nodes except some nodes on the border have three neighbors. A sending node malfunction is recognized by missing or incorrect sequences of signals from one of the neighbors. Therefore, only the directly connected nodes to that faulty node will consider it as faulty. If global knowledge of failure is maintained, then many messages may have to be transmitted on the network since fault can occur frequently which causes high overhead [6].

In a faulty Hex-Cell, it is necessary for message delivery to find a path of non-faulty nodes from source to destination. For this purpose, each node can store some information about its neighbors along with the address of the destination node to help in selecting the proper node for message forwarding. A node in a Hex-Cell network might be in one of the following three states:

1- Normal state: there are no node or link failures.

2- Faulty state: the node is down or the link to reach that node is broken. This is recognized when the neighbors of node don't receive the periodic update from this node for a specific period of time.

3- Dead end state: the node has failure in the incident links which makes it a dead end. This may occur when the node is linked to two dead end nodes, two faulty node neighbors, or one dead end and one faulty node neighbor. A node on the border of a Hex-Cell topology is considered as dead end if one of its neighbors is either faulty or dead end.

FTRH directs the routing decision of a node based on the status of its neighbor. A node will not send messages to nodes which are in dead end or faulty states. A node sends a message indicating that it is in a dead end state, if it has a link with just one other node, or has links with other dead end nodes and another link with a non-dead-end node. FTRH does not allow backtracking. Hence, the message will not be sent back to a node that has forwarded it to prevent loop occurrence which may be checked by inspecting the Come-from parameter. FTRH restricts the number of failure components that can be dealt with to the number of levels. For example if we are at level three, the number of failure components is limited to three.

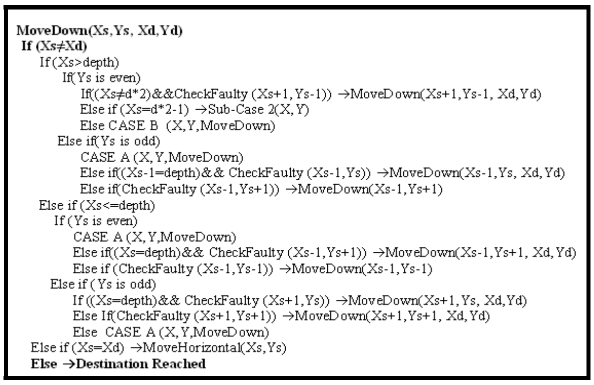
4.2 FTRH Algorithm

Fig. 4 outlines the Fault Tolerance Routing Algorithm for the Hex-Cell network based on the addressing scheme introduced in [1]. The abovementioned three cases are resolved as follows:

<u>Case 1</u>: (Xs<Xd) → MoveDown(Xs,Ys, Xd,Yd) as in Fig. 4(a) - we have here two directions, MoveUp/Left-to-Right, and MoveUp/Right-to-Left.

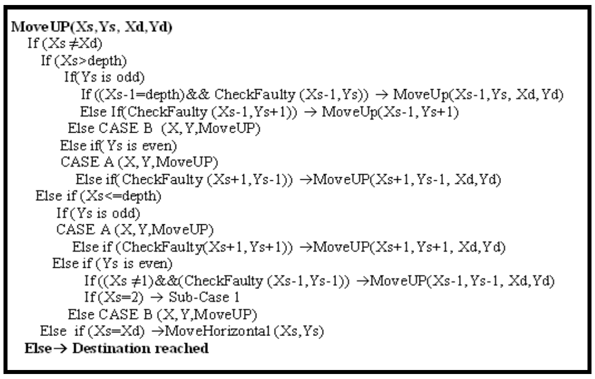
<u>Case 2:</u> (Xs>Xd) →MoveUp (Xs,Ys, Xd,Yd) as in Fig. 4(b) - we have here two directions, MoveDown/Left-to-Right, and MoveDown/Right-to-Left.

<u>Case 3:</u> (Xs=Xd) →MoveHorizontal. (Xs,Ys, Xd,Yd) as in Fig. 4(c) - we have here two directions, MoveHorizontal/Left-to-Right, and MoveHorizontal/ Right-to-Left.



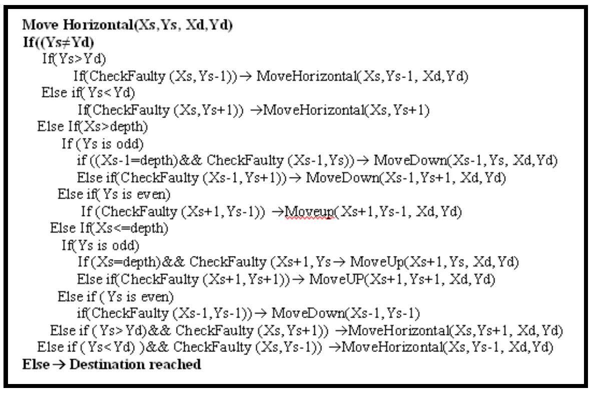
```
MoveDown(Xs,Ys, Xd,Yd)
 If (Xs≠Xd)
    If(Xs>depth)
       If(Ys is even)
          If((Xs≠d*2)&&CheckFaulty (Xs+1,Ys-1)) →MoveDown(Xs+1,Ys-1, Xd,Yd)
          Else if (Xs=d*2-1) →Sub-Case 2(X,Y)
          Else CASE B  (X,Y,MoveDown)
      Else if(Ys is odd)
          CASE A (X,Y,MoveDown)
          Else if((Xs-1=depth)&& CheckFaulty (Xs-1,Ys)) →MoveDown(Xs-1,Ys, Xd,Yd)
          Else if(CheckFaulty (Xs-1,Ys+1)) →MoveDown(Xs-1,Ys+1)
    Else if (Xs<=depth)
      If (Ys is even)
         CASE A (X,Y,MoveDown)
         Else if((Xs=depth)&& CheckFaulty (Xs-1,Ys+1)) →MoveDown(Xs-1,Ys+1, Xd,Yd)
         Else if (CheckFaulty (Xs-1,Ys-1)) →MoveDown(Xs-1,Ys-1)
     Else if ( Ys is odd)
         If ((Xs=depth)&& CheckFaulty (Xs+1,Ys)) →MoveDown(Xs+1,Ys, Xd,Yd)
         Else If(CheckFaulty (Xs+1,Ys+1)) →MoveDown(Xs+1,Ys+1, Xd,Yd)
         Else  CASE A (X,Y,MoveDown)
  Else if (Xs=Xd) →MoveHorizontal(Xs,Ys)
  Else →Destination Reached
```

(a) Case 1: MoveDown.



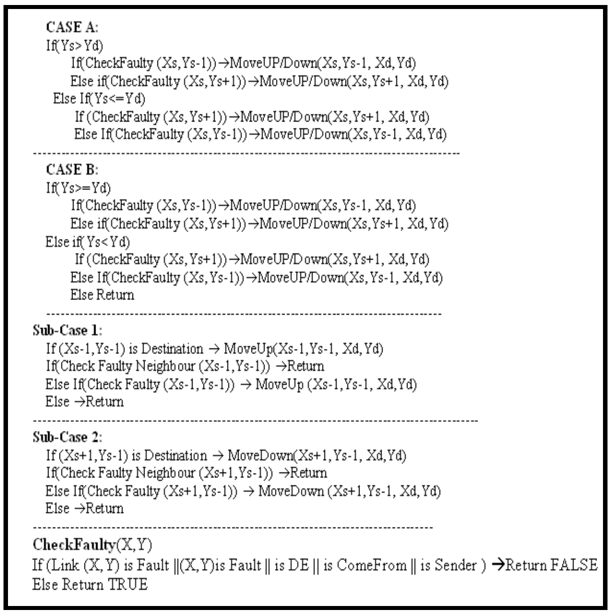
```
MoveUP(Xs,Ys, Xd,Yd)
  If (Xs ≠Xd)
    If (Xs>depth)
       If(Ys is odd)
          If ((Xs-1=depth)&& CheckFaulty (Xs-1,Ys)) → MoveUp(Xs-1,Ys, Xd,Yd)
          Else If(CheckFaulty (Xs-1,Ys+1)) → MoveUp(Xs-1,Ys+1)
        Else CASE B  (X,Y,MoveUP)
      Else if( Ys is even)
      CASE A (X,Y,MoveUP)
         Else if( CheckFaulty (Xs+1,Ys-1)) →MoveUP(Xs+1,Ys-1, Xd,Yd)
   Else if (Xs<=depth)
      If (Ys is odd)
      CASE A (X,Y,MoveUP)
         Else if (CheckFaulty(Xs+1,Ys+1)) →MoveUP(Xs+1,Ys+1, Xd,Yd)
     Else if (Ys is even)
          If((Xs ≠1)&&(CheckFaulty (Xs-1,Ys-1)) →MoveUP(Xs-1,Ys-1, Xd,Yd)
          If(Xs=2) → Sub-Case 1
        Else CASE B (X,Y,MoveUP)
   Else  if (Xs=Xd) →MoveHorizontal (Xs,Ys)
 Else→ Destination reached
```

(b) Case 2. MoveUp.



```
Move Horizontal(Xs,Ys, Xd,Yd)
If((Ys≠Yd)
    If(Ys>Yd)
         If(CheckFaulty (Xs,Ys-1))→ MoveHorizontal(Xs,Ys-1, Xd,Yd)
    Else if( Ys<Yd)
         If(CheckFaulty (Xs,Ys+1)) →MoveHorizontal(Xs,Ys+1)
   Else If(Xs>depth)
      If (Ys is odd)
         if ((Xs-1=depth)&& CheckFaulty (Xs-1,Ys))→ MoveDown(Xs-1,Ys, Xd,Yd)
         Else if(CheckFaulty (Xs-1,Ys+1))→ MoveDown(Xs-1,Ys+1, Xd,Yd)
     Else if( Ys is even)
         If (CheckFaulty (Xs+1,Ys-1)) →Moveup( Xs+1,Ys-1, Xd,Yd)
    Else If(Xs<=depth)
      If(Ys is odd)
         If (Xs=depth)&& CheckFaulty (Xs+1,Ys→ MoveUp(Xs+1,Ys, Xd,Yd)
         Else if(CheckFaulty (Xs+1,Ys+1))→ MoveUP(Xs+1,Ys+1, Xd,Yd)
     Else if ( Ys is even)
         if(CheckFaulty (Xs-1,Ys-1))→ MoveDown(Xs-1,Ys-1)
    Else if ( Ys> Yd)&& CheckFaulty (Xs,Ys+1)) →MoveHorizontal(Xs,Ys+1, Xd,Yd)
   Else if ( Ys< Yd) )&& CheckFaulty (Xs,Ys-1)) →MoveHorizontal(Xs,Ys-1, Xd,Yd)
Else→ Destination reached
```

(c) Case 3. MoveHorizontally.



```
CASE A:
If(Ys>Yd)
      If(CheckFaulty (Xs,Ys-1))→MoveUP/Down(Xs,Ys-1, Xd,Yd)
      Else if(CheckFaulty (Xs,Ys+1))→MoveUP/Down(Xs,Ys+1, Xd,Yd)
 Else If(Ys<=Yd)
       If (CheckFaulty (Xs,Ys+1))→MoveUP/Down(Xs,Ys+1, Xd,Yd)
       Else If(CheckFaulty (Xs,Ys-1))→MoveUP/Down(Xs,Ys-1, Xd,Yd)
-----------------------------------------------------------------------------------
CASE B:
If(Ys>=Yd)
      If(CheckFaulty (Xs,Ys-1))→MoveUP/Down(Xs,Ys-1, Xd,Yd)
      Else if(CheckFaulty (Xs,Ys+1))→MoveUP/Down(Xs,Ys+1, Xd,Yd)
Else if( Ys<Yd)
       If (CheckFaulty (Xs,Ys+1))→MoveUP/Down(Xs,Ys+1, Xd,Yd)
      Else If(CheckFaulty (Xs,Ys-1))→MoveUP/Down(Xs,Ys-1, Xd,Yd)
      Else Return
---------------------------------------------------------------------------------
Sub-Case 1:
  If (Xs-1,Ys-1) is Destination → MoveUp(Xs-1,Ys-1, Xd,Yd)
  If(Check Faulty Neighbour (Xs-1,Ys-1)) →Return
  Else If(Check Faulty (Xs-1,Ys-1)) → MoveUp (Xs-1,Ys-1, Xd,Yd)
  Else →Return
-------------------------------------------------------------------------------------
Sub-Case 2:
  If (Xs+1,Ys-1) is Destination → MoveDown(Xs+1,Ys-1, Xd,Yd)
  If(Check Faulty Neighbour (Xs+1,Ys-1)) →Return
  Else If(Check Faulty (Xs+1,Ys-1)) → MoveDown (Xs+1,Ys-1, Xd,Yd)
  Else →Return
---------------------------------------------------------------------------------
CheckFaulty(X,Y)
If (Link (X,Y) is Fault ||(X,Y)is Fault || is DE || is ComeFrom || is Sender ) →Return FALSE
Else Return TRUE
```

(d) Cases and Subroutines.
Fig. 4 FTRH for HC (d).

The following examples illustrate the proposed algorithm:
<u>Example – Fig. 5: MoveDown Right-to-left</u>Source node: (2, 8), destination node: (5, 3), and faulty nodes: (3, 5), (3, 8), (4, 8).

The algorithm moves down to (2, 8) → (2, 7) → (2, 6) → (2, 5) → (3, 6) → (3, 7) → (4, 7) → (4, 6) → (5, 5) → then moves horizontal from (5, 4) to the destination (5, 3). The nodes (3, 8) and (3, 5) will not be chosen because they are faulty.

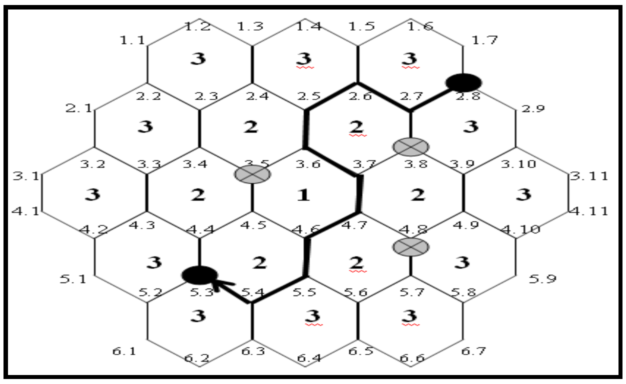


Fig. 5 MoveDown Right-to-left Example.

<u>Example – Fig. 6: MoveUp Left-to-Right with loop free path</u>:
Source node: (3, 7), destination node: (1, 3), and faulty nodes: (1, 4), (3, 6), (2, 3). The algorithm moves up (3, 7) → (3, 8) → (2, 7) → (2, 6) → (2, 5) → (2, 4) → (1, 3).
At node (2, 6), the algorithm will not go to node (1, 5) since it is located at border and one of its neighbors is faulty. This constraint is made in the algorithm to prevent a potential loop on the border.

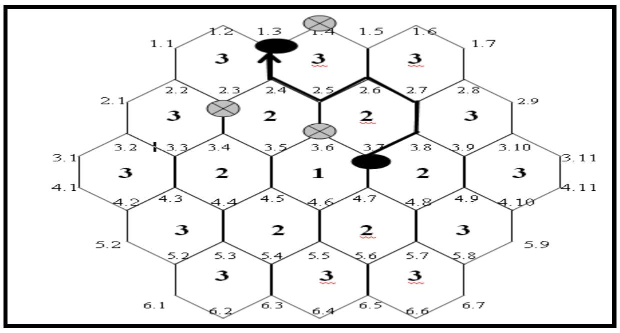


Fig. 6 Move Up Left -to-Right with loop free path Example.

<u>Example – Fig. 7: MoveHorizontal Left-to-Right</u>:
Source node: (5, 1), destination node: (5, 9), faulty nodes: (5, 6), (6, 4), and faulty links: (5, 3)-(5, 4). The algorithm moves horizontally (5, 1)→(5, 2)→(5, 3) then move up to (4, 4)→(4, 5)→(4, 6)→(4, 7)→(4, 8), down to (5, 7) and again moves horizontally from (5, 8) to the destination (5, 9). At node (4, 6) it doesn't go (5, 5) because two of its neighbors are faulty; i.e. it is a dead node.

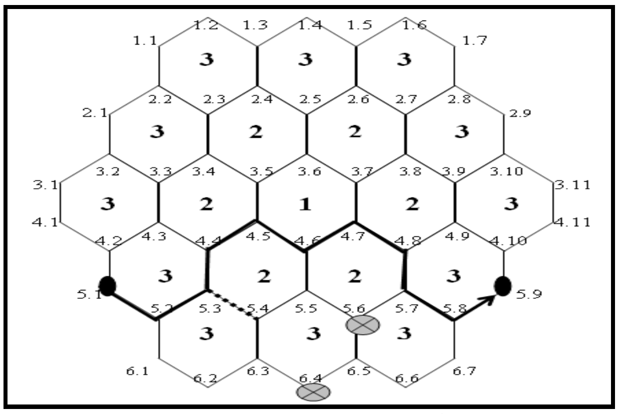


Fig. 7 MoveHorizontal Right-to-left Example.

<u>Example – Fig. 8: MoveDown Left-to-Right with loop free path</u>:Source node:(4, 7), destination node:(5, 5), and faulty nodes:(5, 6), (4, 6), (5, 8). The algorithm moves down (4, 7)→(4, 8)→(4, 9)→(3, 9)→(3, 8)→(3, 7)→(3, 6) →(3, 5)→(4, 5)→(4, 4)→(5, 3). Then moves horizontally to (5, 4)→(5, 5). In this example, at node (4, 8), routing will not go to node (5, 7), since its two direct neighbors are faulty. Also, at node (3, 7) it doesn't go to (4, 7) because it is the original sender of the message.

5. Results and Discussion

In order to have confidence in the results we ran the algorithm for a 120 different sample cases. Each case represents sending a message from a source to a destination with the existence of faulty nodes and/or links. The number of faulty nodes or links is equal to the depth of the Hex-Cell. The 120 sample cases were divided into four groups such that each group represents different depth of the Hex-Cell such as 2, 3, 4, and up to 5 with various sources and destinations. Table 1 shows a comparison of the proposed FTRH algorithm with a hypothetical optimal routing algorithm with respect to the routes lengths. From the table, we can notice that FTRH is identical to the optimal solution for 55 cases out of the 120 test cases. The route length for FTRH is slightly greater than optimal for the remaining cases. These cases tend to appear if all or most of the faulty components are on the same row. The message goes through longer path to reach the destination.

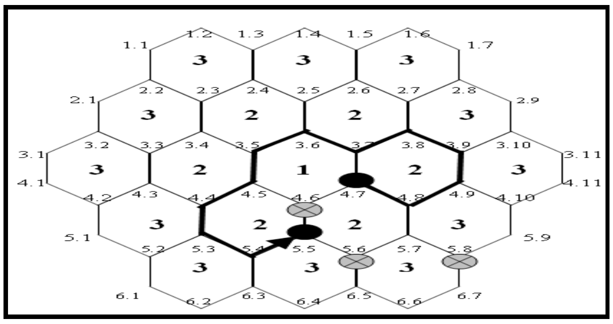


Fig. 8 MoveDown Left -to-Right with a loop free path.

Table 1: Cases and Comparisons

	DEPTH 2					DEPTH 3					DEPTH 4					DEPTH 5			
CASES	**Optimal**	**Fault Tolerant**	**Difference**	**Relative Error**		**Optimal**	**Fault Tolerant**	**Difference**	**Relative Error**		**Optimal**	**Fault Tolerant**	**Difference**	**Relative Error**		**Optimal**	**Fault Tolerant**	**Difference**	**Relative Error**
1	7	7	0	0		9	9	0	0		12	12	0	0		23	23	0	0
2	8	8	0	0		12	12	0	0		13	13	0	0		28	28	0	0
3	9	9	0	0		6	6	0	0		9	9	0	0		10	10	0	0
4	9	9	0	0		7	7	0	0		12	12	0	0		18	18	0	0
5	8	8	0	0		10	10	0	0		14	14	0	0		22	22	0	0
6	7	7	0	0		10	10	0	0		12	12	0	0		12	12	0	0
7	8	8	0	0		11	11	0	0		8	8	0	0		12	12	0	0
8	6	6	0	0		9	9	0	0		12	12	0	0		21	21	0	0
9	5	5	0	0		14	14	0	0		13	13	0	0		10	10	0	0
10	6	6	0	0		8	8	0	0		16	16	0	0		22	22	0	0
11	7	7	0	0		6	6	0	0		17	17	0	0		16	16	0	0
12	9	9	0	0		13	13	0	0		11	11	0	0		12	12	0	0
13	8	8	0	0		14	14	0	0		10	10	0	0		21	22	1	0.5
14	5	5	0	0		10	10	0	0		5	5	0	0		21	23	2	0.10
15	10	10	0	0		15	15	0	0		11	11	0	0		22	24	2	0.09
16	8	9	1	0.13		19	20	1	0.06		13	13	0	0		16	18	2	0.13
17	7	8	1	0.14		13	14	1	0.08		14	15	1	0.07		16	18	2	0.13
18	15	16	1	0.07		9	11	2	0.22		18	19	1	0.06		15	18	3	0.2
19	9	11	1	0.22		11	13	2	0.18		16	17	1	0.06		20	23	3	0.15
20	7	9	2	0.29		5	7	2	0.4		14	16	2	0.14		17	20	3	0.17
21	7	9	2	0.29		11	13	2	0.18		15	17	2	0.13		13	17	4	0.38
22	8	10	2	0.25		17	19	2	0.12		12	14	2	0.17		21	25	4	0.19
23	7	9	2	0.29		10	12	2	0.2		18	20	2	0.11		21	25	4	0.19
24	6	8	2	0.33		8	12	4	0.5		15	17	2	0.13		19	23	4	0.21
25	8	10	2	0.25		8	12	4	0.5		21	23	2	0.10		20	24	4	0.2
26	5	7	2	0.4		9	13	4	0.44		14	18	4	0.29		10	15	5	0.5
27	6	8	2	0.333		7	12	5	0.71		15	19	4	0.27		23	29	6	0.26
28	3	6	3	1		12	14	6	0.17		10	22	12	1.2		12	18	6	0.5
29	6	10	4	0.67		5	15	10	2		5	19	14	2.8		7	19	12	1.716
30	8	12	4	0.5		5	17	12	2.4		5	21	16	3.2		4	22	18	4.5
	Differences Average= 1.03 Relative Error =0.17					**Differences Average= 1.97 Relative Error =0.27**					**Differences Average=2.17 Relative Error = 0.29**					**Differences Average= 2.8 Relative Error = 0.32**			

We may notice that the average difference between the proposed protocol and the hypothetical optimal one is 1.95. As shown in Fig. 9, the average difference in the route length increases as the depth increases, with a reasonable maximum average difference equals to 2.8 for depth = 5.

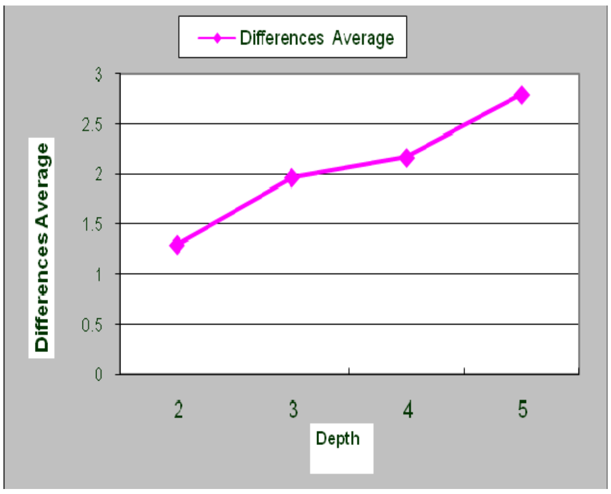


Fig. 9 The differences average between the fault tolerance routing algorithm and a hypothetical optimal algorithm.

Fig. 10 shows the Relative Error with respect to depth of the network which is calculated as:

Relative Error = (FTRH path length - Optimal path length) / Optimal path length

The average of relative error for all cases is 0.26. Fig. 10 confirms that the efficiency of the proposed protocol decreases as the depth increases.

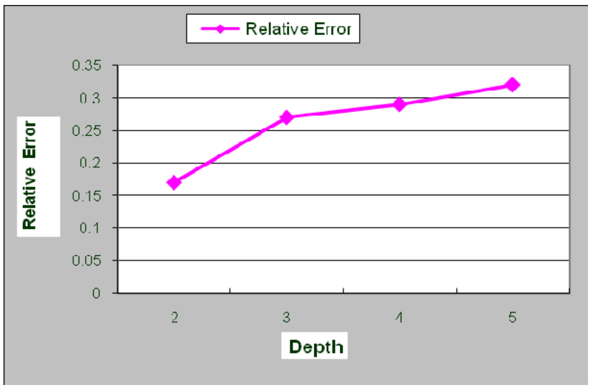


Fig. 10 The Relative Error between FTRH and a hypothetical optimal algorithm.

Fig. 11 describes the average route length in terms of the topology depth for FTRH versus an optimal solution. The Fig. shows that the routes lengths of the two protocols are quite close to each other. Although the average route length increases as the depth increases, the difference between the two protocols slightly increases. The closeness of the proposed algorithm to the optimal one, stresses its effective way of choosing the route around faulty nodes.

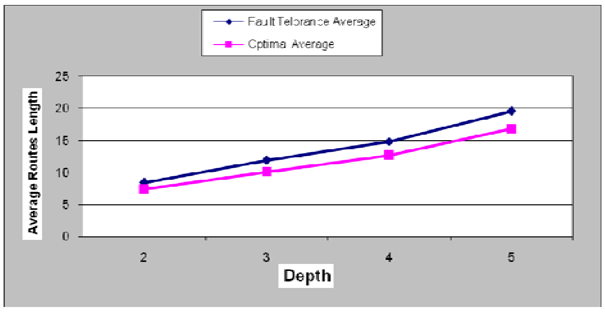


Fig. 11 Average route length versus network depth.

6. Conclusion

This paper introduces a fault tolerance routing algorithm for the Hex-Cell network topology. The algorithm guarantees the delivery of messages even with the presence of component failures. The algorithm is evaluated and compared with a hypothetical optimal fault tolerance algorithm for the length of the resulting route. It produces optimal path lengths for 46% of the cases. A slight increase in the route length is noticed for the remaining studied cases. The closeness of the proposed algorithm to the optimal one increases the reliability of routing process in the Hex-Cell network topology by ensuring the delivery messages to destination even with the presence of failure components.

References

[1] Ahmad Sharieh, Mohammad Qatawneh, Wesam Almobaideen, Azzam Sleit, Hex-Cell: Modeling topological properties and routing algorithm, Euro Journal of Scientific Research, 22(2) (2008) 457 – 468.

[2] Jong-Hoon Youn, Bella Bose, Seungjin Park, Fault-Tolerant routing algorithm in meshes with solid faults, Journal of supercomputing, 37(2006) 161-177.

[3] Dajin Wang, A Low-cost fault-tolerant structure for hypercube, Journal of Supercomputing, 20(2001) 203-216.

[4] Flaviu Cristian, Bob Dancey, Jon Dehn, Fault-tolerance in air traffic control systems, Transactions on Computer Systems (TOCS), 14(3) (1996) 265-286.

[5] Ivan Stojmenovic, Honeycomb Networks: Topological properties and communication algorithms, IEEE Transactions on Parallel and Distributed Systems, 8(10) (1997) 1036-1042.

[6] Fabian Garcia Nocetti, Ivan Stojmenovic, Jingyuan Zhang, Addressing and routing in hexagonal networks with applications for tracking mobile users and connection rerouting in cellular networks, IEEE Transaction on Parallel Distributed Systems, 13(9) (2002) 963-971.

[7] A.Mejia, J. Flich, J. Duato, Sven-Arne Reinemo, Tor Skeie, Segment-based routing: An efficient fault-tolerance routing algorithm for mesh and tori, Proceedings of the 20th IEEE International Parallel and Distributed Symposium, 2006.
[8] Mahmoud Al-Omari, Mohammed Mahafzah, Fault-tolerant routing in hypercubes using masked interval routing scheme, Proceedings of the 1999 ACM symposium on Applied computing, 481-485.
[9] Huaxi Gu , Jie Zhang, Zengji Liu, Xiaoxing TU, Routing in hexagonal networks under a corner-based addressing schema, The institute of Electronics, Information and Communication Engineers, 2006.
[10] Suresh Chalasani, Rajendra V. Boppana, Fault-tolerant wormhole routing in Tori, International Conference on Supercomputing, (1994) 146-155.
[11] Catherine Decayeux, David Seme, 3D hexagonal network: modeling, topological properties, addressing scheme, and optimal routing algorithm, IEEE Transactions on Parallel and Systems, 16(9) (2005), 875-884.
[12] Qatawneh Mohammad, Adaptive fault tolerant routing algorithm for Tree-Hypercube multicomputer, Journal of Computer Science, 2(2) (2006) 124-126.
[13] Jipeng Zhou and Francis C.M. Lau, Multi-phase minimal fault-tolerant wormhole routing in meshes, Parallel Computing. 30 (3) (2004), 423–442
[14] K. Day K, S. Harous, A. Al-Ayyoub, A Fault tolerant routing scheme for Hypercubes, Telecommunication Systems, 13(1) (2000) 29-44.
[15] J. Al-Sadi, K. Day and M. Ould-Khaoua, Fault-tolerant routing in hypercubes using probability vectors, Parallel Computing, 27(10) (2001) 1381-1399.

Mohammed Qatawneh is an associate professor and Chairman of the Computer Science Department at the University of Jordan. He received his Ph.D. in Computer Engineering from Kiev University in 1996. Dr. Qatawneh published several papers in the areas of parallel algorithms, Networks and Embedding systems. His research interests include Parallel Computing, Embedding System, and Network Security.

Wesam AlMobaideen received his Ph.D. in computer Networks from the University of Bologna, Bologna Italy in 2003, Master degree in Computer Science from the University of Jordan in 1999 and B.Sc. degree in Computer Science from Mu'ta University, Karak, Jordan in 1997. He is currently an sssociate professor with the Computer Science Department at the University of Jordan. He held several administrative positions including the Assistant Dean of King Abdullah II School for Information Technology and Head of Quality Assurance.

Azzam Sleit is an associate professor with the Computer Science Department and the Director of the Computer Center, University of Jordan. His research interests include imaging databases, information retrieval, and distributed systems. Before Joining the University of Jordan in 2005, Dr. Sleit was the Chief Information Officer at Hamad Medical / Qatar's Ministry of Public Health where he developed and executed the information technology strategy of healthcare for State of Qatar. Dr. Sleit has over sixteen years of experience and leadership in the information technology field including work at all levels of government, private and international sectors. Before joining Hamad Medical, Dr. Sleit was the Vice President of Strategic Group & Director of Professional Services of Triada, USA. Dr. Sleit served as the Director of Professional Services at Information Builders, USA. Dr. Sleit holds B.Sc., M.Sc. and Ph.D. in Computer Science from Wayne State University, USA.