Fault-Tolerant Message Routing for Multiprocessors

Lev Zakrevski, Mark Karpovsky[♣]

Research Lab. on Reliable Computing,
Boston University, Department of Computer Engineering,
8 Saint Mary's St., Boston, MA 02215, USA
zakr@bu.edu, mr@enga.bu.edu

Abstract. *In this paper the problem of fault-tolerant message routing in two-dimensional meshes, with each inner node having 4 neighbors, is investigated. It is assumed that some nodes/links can be faulty, so it is necessary to route messages, using local information at each step. A new and efficient algorithm is proposed to solve this problem. This algorithm is local and consists of pre-routing and routing stages. The pre-routing algorithm is implemented off-line. The complexity of the pre-routing stage is O(tN), where N is the number of nodes in the system, and t is the number of faulty nodes. The complexity of the on-line routing stage (the size of the routing table stored in the local memory) is O(t). The pre-routing algorithm is performed only once, after a new fault is detected. The algorithm allows 100% of deliverable messages to be delivered in the presence of faulty nodes with no deadlocks or lifelocks. No nodes are declared unsafe. The main idea is to construct fault-free rectangular clusters during the pre-routing stage and store the information about their boundaries in local memories. At the routing stage the direction for sending a message at any node is determined by a cluster to which the destination node belongs. The algorithm is generalized on the case of multidimensional meshes.*

Key Words. Fault-tolerant network computing, multiprocessors, meshes, routing algorithms, adaptive routing

1 Introduction

The problem of fault-tolerant message routing is of a great practical importance. It is assumed that some of the nodes and/or links in the computer network can be faulty, and it is necessary to find, if possible, a simple routing algorithm for message transmission between non-faulty nodes. Due to the large number of nodes, the routing algorithm must be local, i.e. that no central host will be used for message routing.

[♣] This work was supported by the NSF under Grant MIP 9630096

The above problem was investigated by several authors (see e.g. [1-5]). The common drawback of the existing methods is that a message from a source to a target node will not always find its way, even if it exists. This means, that a large number of potentially “unsafe” nodes, which are excluded from communication may be present in the network. In this paper we propose an adaptive routing algorithm which does not require declaring any fault-free nodes unsafe. If it is possible for a message to be delivered from node *S* to node *D*, then the corresponding path will be found by the proposed algorithm.

Suppose that we have a multiprocessor network, consisting of *N* nodes, each of these (except of the border ones) is connected to 2*n* neighbours. In this paper we will analyze first the case of two-dimensional meshes with $r\times r$ nodes, where each inner node has 4 neighbours. The proposed methods are expanded to other regular structures, such as two-dimensional torus and three-dimensional meshes.

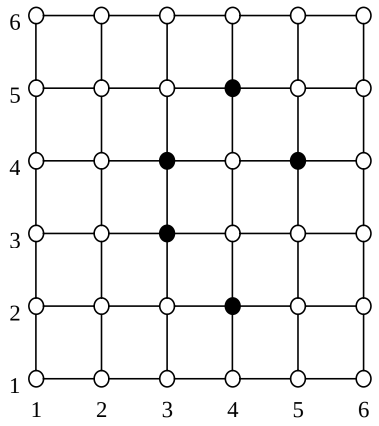

Fig. 1. An example of the 2-dimensional mesh with faulty nodes shaded

The maximal number of faulty nodes will be denoted by *t*. Unless specifically stated otherwise, we will assume that only nodes (and not links) are faulty. In Section 5 we discuss the case when links can be faulty. Each node can test its neighbors. We assume that if a testing node is fault-free, it will receive the correct state of the node under test (including the link between these nodes). Only catastrophical (permanent) faults are investigated. An example of an 6×6 network with t=5 faulty nodes is shown in Fig.1. (Faulty nodes in Fig.1 and following Figs are 7shown in black color.)

In Section 2 an overview of the existing methods for fault-tolerant routing in meshes is given. Criteria for comparison between routing algorithms are introduced and necessary requirements are formulated.

In Section 3 a new adaptive cluster algorithm for fault-tolerant message routing in 2-dimensional meshes is described. The algorithm consists of the following steps:

- off-line pre-routing (at the end of this stage information will be stored in the routing table of each node. This information is used for message routing);

- on-line message routing during the normal work of the network. At this stage the message routing algorithm has to decide:

1) whether the message will be sent;

2) if it will be sent, then what direction will be used for routing.

Section 4 is devoted to generalizations of the proposed algorithms. In particular, 2-dimensional toruses and 3-dimensional meshes are analyzed. A more general fault model, including dynamic faults and faulty links, is considered.

Experimental results are given in Section 5. The simulation results illustrate the efficiency of the proposed algorithm for 2-dimensional meshes.

2 Overview of the Existing Algorithms

Different approaches to the problem of constructing message routing algorithms in networks containing faulty nodes are outlined below. They can be compared using such criteria as:

1) reliability (the probability that message will not be delivered after sending or the probability that a message to a reachable destination will not be sent);

2) dilation time - additional length of a message path compared with the shortest possible path;

3) the number of necessary computations - related to 2);

4) amount of additional information stored in a message;

5) amount of additional information stored in one node (size of the routing table);

6) length of the initial pre-routing stage;

7) the complexity of information update after new faults occur.

To be efficient, an algorithm has to be deadlock-free and lifelock-free. It means that any message will be either delivered, or at least efficiently canceled. Cycles must be avoided. It is essential for the case of many messages in the system [4,10].

All routing algorithms can be divided into exact and non-exact algorithms. For an exact algorithm we can be sure that if a message is sent, it will reach the destination. A non-exact algorithm can cancel a message. A simple example of such an algorithm is a directional one - if a message can not be sent one step closer to destination, it will be canceled.

Any non-exact algorithm can be transformed into exact one, if during the pre-routing stage messages with all possible sources and destinations are sent, acknowledgments received and this global information stored in local memories. This approach is time-consuming and depending on the algorithm which is used at the pre-routing stage for message transmission, there can be a situation when a message will not be sent although in principle it can be delivered.

An example of the deadlock-free routing algorithm is NAFTA [2,3]. This algorithm is based on the combination of North-Last and South-Last strategies and allows deadlock-free routing without global knowledge. However, it introduces some unsafe nodes. The number of these nodes can grow as $O(t^2)$ for the worst configuration of faulty nodes.

Another recent algorithm [6] uses the concept of faulty rings to route around faulty nodes. Its main drawback is severe restrictions on fault locations - it is prohibited to have faulty nodes both to the *South* and *North*, or to the *East* and *West*, from a fault-free node.

The main idea of the proposed cluster algorithm consists of the construction of fault-free clusters (rectangles) at the off-line pre-routing stage. Within fault-free rectangles a simple deadlock-free algorithm can be used for message transfers. A cluster connectivity graph is also formed at the pre-routing stage to be used for routing. For each message being sent, first a path in the cluster terms is determined, and then the message is sent to the corresponding adjacent cluster. This will minimize the computational complexity at the routing stage at the price of additional work at the pre-routing stage. The pre-routing algorithm is performed only once after new faults are detected. The size of a local memory, needed to store the routing information, will decrease from the number of nodes N to the number of clusters $C = O(t)$, where t is the number of faults.

The proposed algorithm allows to achieve an efficient routing in the presence of a large number of faults. In particular, it is scaleable - we can increase N, keeping t/N constant. As opposed to the approaches presented in [2,3], the proposed algorithm does not require that some fault-free nodes be declared “unsafe” and excluded from the system.

3 Adaptive Routing Using Fault-Free Clusters

At the off-line *pre-routing* stage, the following operations have to be implemented:

1) Cluster’s formation.

2) The boundaries of all clusters (fault-free rectangles) are broadcasted to all reachable fault-free nodes. In each of these nodes a list of all reachable clusters is constructed. If several clusters are identical, only one of them remains in the list.

3) For each cluster, adjacent clusters are determined.

4) A cluster connectivity graph is formed and the shortest path problem for this graph is solved. As the result at this stage, routing tables are constructed and stored in local memories. (Contents of these tables will be different for different nodes.)

At the *routing* stage for each message it is determined, whether the destination node is reachable, and if it is, to which cluster it belongs. After this the routing table is used to find the next cluster on the path to the destination cluster.

3.1 Cluster Formation

The main concept, used in the proposed algorithm, is the concept of a cluster. Only rectangular clusters are considered in this paper, to minimize the memory requirements to store the information about them.

Definition 1. A cluster is defined as a rectangular area in the mesh, free from faulty nodes and links.

Each cluster is formed by the expansion first in *x* than in *y* direction starting with one fault-free “basic” node. The following nodes are selected as basic ones, if they are fault-free:

- nodes to the *North*, *West*, *East* from the corresponding faulty nodes,

- *South-West* corner node of the mesh, with the coordinates (1,1) (or any one node for the toroid structure).

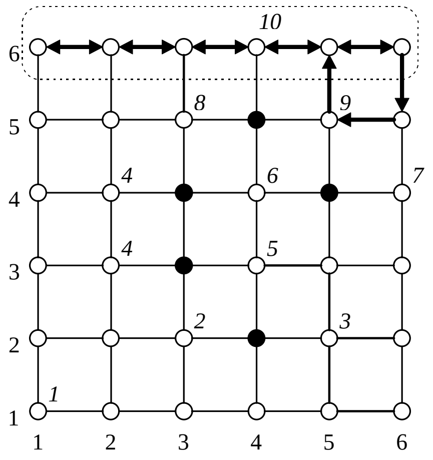

Fig. 2. Basic node selection and message routing during cluster formation

Basic nodes (*1*,*2*,...,*10*) for the example of Fig.1 are presented at Fig.2. In this case we have *NB*=11 basic nodes and *C*=10 clusters (two basic nodes generate the same cluster).

To construct a cluster, the (*x*,*y*)-expansion algorithm (first - expansion along the row, then - along the column), beginning from the basic nodes, is used. (The example of cluster formation for basic node *10* with coordinates (4,6) is represented in Fig.2). Clusters can intersect. If a node is faulty, all messages from this node will be ignored by the fault-free nodes.

Let us analyze the process of cluster formation in detail, taking basic node *10* with coordinates (4,6) as an example (see Fig.2). At the first stage, two messages are sent *East* and *West*, until they come to the edges of the mesh (*East*, node (6,6) and *West*, node (1,6)), then they return back, bringing *left_edge* and *right_edge* parameters (1 and 6) to the basic node *10*. At the second stage, two messages are sent from the basic node of the rectangle under construction one more time, one traveling to the top, another to the bottom of the network. The second message, for example, will travel the following path: *East* to the *right_edge*, one node to the *South*, *West* to the *left_edge*, one node to the *South*, etc., until a fault or an edge of the mesh is reached. After this the message travels back to the basic node. On the forward path the coordinates of the basic node, *left_edge* and *right_edge* are transmitted, at the backtracking step *top_edge* is sent to the basic node. When both *top_edge* and *bottom_edge* parameters are received by the basic node, the cluster is constructed. (For the above example *top_edge* = *bottom_edge*=6. The constructed cluster is shown at Fig.2 by the dot line)

The maximal complexity in hops of this stage is of the order $t\times r^2$, but the cluster formation can be implemented in parallel by a local algorithm, reducing the time complexity for this stage. We note that at the cluster formation stage each node from a row containing a basic node, receives not more than 4 messages, and all other nodes - not more than 2.

It is worth mentioning also that the process of cluster formation for several basic nodes can result in the same cluster. In this case only one cluster remains in list. This elimination is done at the next stage, during the cluster list formation. To reduce the number of generated clusters the following rule is used during cluster formation - if to the *West* of the given basic node there is another basic node, the process of cluster formation is aborted.

For the above example the following 10 different clusters will be constructed:

[***1***] = (1,6,1,1); [***2***] = (1,3,1,2); [***3***] = (5,6,1,3); [***4***] = (1,2,1,6); [***5***]= (5,6,3,3);
[***6***] = (4,4,3,4); [***7***] = (6,6,1,6); [***8***] = (1,3,5,6); [***9***] = (5,6,5,6); [***10***]=(1,6,6,6).

Here (x_1, x_2, y_1, y_2) are boundaries of a cluster $(x_1\le x_2, y_1\le y_2)$. The total number of nodes in a cluster is $(x_2\text{-}x_1+1)\times(y_2\text{-}y_1+1)$. Cluster [***i***] is generated by the basic node *i*.

Remark. If the number of faults *t* is greater than *r*, then one can select as basic all fault-free nodes from the first column and to the *East* from the faulty nodes. Also in this case it is advantageous to modify the process of cluster construction. Namely, clusters will be constructed in 2 steps. At the first step for each fault-free node the distances to the closest faulty node (or to the edge of the mesh) to the *North* and to the *South* are determined. At the second step, for each basic node only *x*-expansion is made and minimal distances of the included nodes are used to determine north and south boundaries of the constructed cluster. In this case complexities of cluster formation and cluster broadcasting are decreasing (see Section 2.2).

Since the number of clusters *C* is important to estimate a complexity of the algorithm, we will present below some upperbounds on *C*.

Theorem 1. For a 2-dimensional $(r\times r)$ mesh with *t* faulty nodes the number of clusters is upperbounded by

$$C \le min\,(3t+1,\, t+r,\, \lceil r^2/2\rceil). \qquad (1)$$

Proof. **1**. Since each basic node corresponds to a cluster, and each cluster corresponds to at least one basic node, we have from the definition of basic nodes $C \le 3t+1$.

2. All clusters can be classified by the corresponding basic nodes into *x*-clusters, which are formed by the basic nodes, which are to the *East/West* from the fault, and *y*-clusters, formed by *North* basic nodes and node (1,1). Only *y*-clusters that do not coincide with *x*-clusters are required. Let us suppose that there are *l* fault-free rows in the mesh. Then the number of different *y*-clusters is upperbounded by *l*, since any two basic nodes in a fault-free row will form the same cluster. If there is a fault either to the west or to the east from a basic node, which is to the *North* from another faulty node, then the corresponding y-cluster will be the same as one of the x-clusters. A number of different x-clusters, generated by the basic nodes in a row with t_i faults is upperbounded by $t_i + 1$. Making a sum over all such rows, we receive $t+r-l$. It means that the total number of clusters is upperbounded by $t+r$.

3. To prove the last part of the statement, it is sufficient to note that for each row the number of different x-clusters is upperbounded by $\lceil r/2 \rceil$, and for each two adjacent rows - by r.

We note that all three upper bounds of Theorem 1 are precise, i.e. it is possible to construct fault patterns for each one of the bounds such that numbers of clusters for these fault patterns are equal to the corresponding bounds in (1). Thus we have from Theorem 1 for the compression rate C/N for global information to be stored in local memories

$$C/N \le \begin{cases} (3t+1)/r^2, & t \le r/2; \\ t/r^2 + 1/r, & r/2 < t \le r^2/2; \\ 0.5, & t > r^2/2. \end{cases} \qquad (2)$$

Theorem 2. For a 2-dimensional ($r \times r$) mesh with t faulty nodes each fault-free node of the mesh belongs to at least one and at most $t+1$ of the clusters.

Proof. **1**. Let us suppose that there exists a fault-free node which does not belong to any cluster. Consider a southmost such node. There are two cases - when it belongs to a fault-free row or not. In the second case either to the left or to the right of the node (perhaps after some fault-free nodes) a basic node can be found. In the first case it is either a bottom row, containing a fault-free node (which is covered by basic node (1,1)), or a node just *South* belongs to a cluster, which can be expanded one line to the *North*. This contradicts the original assumption.

2. To prove that the maximal number of clusters, to which a given node A belongs, is upperbounded by $t+1$ recursion by t can be used. Let us assume that this statement is valid for $t=k$ and prove it for $t=k+1$. An arbitrary faulty node B can be considered "additional" ($k+1$)-th faulty node. For each faulty node, clusters containing nodes to the *East* and *West* from this faulty node, can not intersect, the same is true for the *North/South* nodes. It means that node A belongs to at most two clusters, generated by fault B. However, if A belongs to two such clusters, then in the absence of fault B nodes A and B belong to the same cluster, now deleted. So the total number of clusters, intersecting at one node, is upperbounded by $(k+1)-1+2= t+1$. If $t<r-1$, this bound can not be improved, as it is clear from the example given at Fig.3, where $t=4$ and node (1,1) belongs to 5 clusters. (For this case, $N=36$, $r=6$, $C=10$).

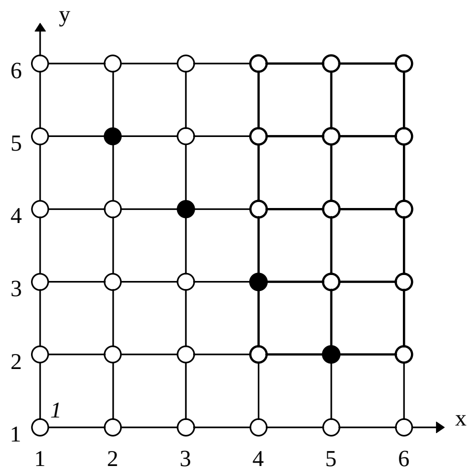

Fig. 3. The configuration of faulty nodes such that a fault-free node (1,1) belongs to t+1=5 clusters

We note also that if several faults can be grouped in one rectangle, then the number of basic nodes and the number of clusters can be reduced. Namely, instead of t in the formulation of the Theorems 1 and 2 the number d of disjoint rectangles, covering faulty nodes, may be used. Also, the basic nodes in this case can be selected in the following way: for each faulty rectangle, 3 basic nodes are generated - the southmost of the *East* and *West* of faulty nodes in the faulty rectangle and the westmost of the *North* nodes (if these nodes are fault-free). So, the total number of basic nodes is upperbounded by 3d+1. (An example is given at Fig. 2 where faults (3,3) and (3,4) form the rectangle, so instead of 2 basic nodes - (2,3) and (2,4) - only node (2,3) is sufficient.)

3.2 Broadcasting of Clusters Boundaries in 2-Dimensional Meshes with Faulty Nodes

After cluster formation each basic node has an information about the boundaries of the corresponding cluster generated by this node. This information must be transmitted to other nodes to form a cluster list, which will be stored in all nodes (local memory of size $O(t)$ will be required for this). To this purpose the following simple broadcasting algorithm is used.

Let us assume that a message M need to be transmitted from the basic node S to all other reachable nodes. To do this, each node must perform the following procedure: when it receives M, it checks, whether this information is already stored at the local memory, if yes, nothing is made. If the received information is new, message M is transmitted to the 3 other directions (excluding faulty nodes). In such a way, each node can generate not more than 3 messages (any basic node can have maximum 3 non-faulty neighbors).

We note that the total number of messages for broadcasting from one basic node to all nodes is not more than 3N. Each node receives not more than 4 messages.

At the cluster list formation process each basic node generate a message, containing its coordinates and 4 numbers, defining the rectangle borders - all together 6 log_2r bits. To minimize the total number of messages, one additional improvement is used - each time when a new message is received, it is checked, whether in the local memory exists information about another cluster with the same borders. In this case only the cluster with the minimal basic node is left in the list; if it is a new one, the message will be transmitted to the neighbors.

This "storing forward" broadcasting can be implemented efficiently and without deadlocks if at the broadcasting stage we allocate in each one of the local memories for every source (basic node) a block of size 6 log_2r. Since by Theorem 1 a number of basic nodes is upperbounded by $3t$+1, $3t$+1 blocks will be sufficient.

It is well known that storing forward broadcasting may be very inefficient but in our case this approach will be sufficient for the following reasons:

1. This broadcasting is implemented off-line and only when a new fault in the system is detected.
2. Every source is sending a single message.
3. These messages are short (6 log_2r information bits).
4. A number of sources is small (at most $3t$+1 or at most $t+r$).

We note that for the described procedure some clusters can be completely covered by a union of other clusters. In general, such clusters can be removed from the list but this cluster reduction is difficult to implement (the covering problem is NP-hard [14]).

3.3 Cluster Connectivity Graphs

At the next step in each node the same cluster connectivity graph is formed and the shortest path problem for this graph is solved, using a well-known algorithm for a weighted shortest path in a non-oriented graph, (see e.g. [15]). This procedure is implemented independently in each node, using a shortest distance (in links) from this node to a destination as a weight. The computational complexity of this procedure is $O(t^2)$, no hops are needed.

The cluster connectivity graph for the above example is shown at Fig.4.

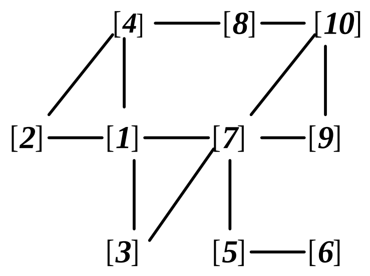

Fig.4. The cluster connectivity graph for the mesh of Fig.1

The whole graph does not need to be stored. All that is needed is to store in each node the name of the next cluster for the shortest path, the distance from the node to the cluster and the coordinates of the closest belonging to this cluster node, called entry node - arrays ***NC***, ***D*** and ***K*** are used for this purpose, where ***NC(i)*** is the number of the next cluster, if [***i***] is a destination cluster, ***D(i)*** - distance (in links) from the source to the cluster [***i***] (***D***(*i*) = 0 if a node under consideration belongs to cluster [***i***]), ***K(i)*** - closest to the source node from cluster [***i***] (entry node).

To construct these arrays, a modified search in breadth is used. We use 2 node lists - *open* and *close*, initially first have all clusters that contain the node under consideration, and the second one is empty. During each step the cluster is selected from *open*, which has a minimal distance from the source node (minimum ***D(i)***). Selected cluster is moved to *close*, and all new clusters, adjacent to it, are added to *open*. If cluster [***x***] is moved and cluster [***y***] added, then ***NC(y)*** is set as ***NC(x)***, if ***NC(x)***$\neq$ 0, otherwise ***NC(y)***=*x*. Also if cluster [***y***] is adjacent to [***x***] and belongs to the *open* already, it is checked whether a new distance (via cluster [***x***]) from the source to [***y***] is less than the existing one; if yes, then values of ***D(y)*** and ***K(y)*** are modified. New value of *D(y)* is calculated as *D(x)+R*, where *R* is the distance between nodes *K(x)* and *K(y)*, *K(y)* is the nearest to *K(x)* node in cluster [*y*]. In a similar way arrays ***D*** and ***K*** are constructed. If a source node (*a*,*b*) belongs to the cluster [***i***], then ***NC(i)***=***D(i)***=0, ***K(i)***=(*a*,*b*). The process always begins from the cluster [***j***] that has ***D(j)***=0 (there can be several such clusters that contain the initial node under consideration).

To illustrate the above algorithm, let us consider a node (5,3) at Fig. 2. After a pre-routing stage, it will have the following information stored in its local memory:

(1) array ***B*** [16,6] (*t*=5), containing the information about the basic nodes and the borders of the corresponding clusters

B [***1***] = (1,1,1,6,1,1);	***B*** [***2***] = (3,2,1,3,1,2);	***B*** [***3***] = (5,2,5,6,1,2);
B [***4***] = (2,3,1,2,1,6);	***B*** [***5***] = (4,3,5,6,3,3);	***B*** [***6***] = (4,4,4,4,3,4);
B [***7***] = (6,4,6,6,1,6);	***B*** [***8***] = (3,5,1,3,5,6);	***B*** [***9***] = (5,55,6,5,6);
B [***10***] = (4,6,1,6,6,6);	***B*** [***11***] = (0,0,0,0,0,0); ...	

where ***B***[***i***]= (i_x, i_y, x_1, x_2, y_1, y_2), i_x, i_y are coordinates of the basic node *i*, x_1, x_2 and y_1, y_2 are *x* and *y* coordinates for boundaries of the cluster [***i***].

(2) array ***NC*** [16] (only first 10 values will be used)

NC(1)= 1, ***NC***(2)= 1, ***NC***(3)=0, ***NC***(4)= 1, ***NC***(5)= 0,
NC(6)= 6, ***NC***(7)= 7, ***NC***(8)=7, ***NC***(9)= 7, ***NC***(10)= 7,

(3) array ***D*** [16] (only first 10 values will be used)

D(1)= 2, ***D***(2)= 4, ***D***(3)=0, ***D***(4)= 5, *D*(5)= 0,
D(6)= 1, ***D***(7)= 1, ***D***(8)=7, ***D***(9)= 3, ***D***(10)= 4,

(4) array ***K*** [16] (only first 10 values will be used)

K(1)= (5,1), ***K***(2)= (3,1), ***K***(3)=(5,1), ***K***(4)= (2,1), ***K***(5)= (5,1),
K(6)= (4,3), ***K***(7)= (6,3), ***K***(8)=(3,6), ***K***(9)= (6,5), ***K***(10)= (6,6),

The node (5,3) belongs to the clusters [***3***] and [***5***]. So, these two clusters correspond to the first two opened nodes for the process of the cluster connectivity graph construction.

3.4 On-Line Routing Stage

At the *routing* stage for each message a destination cluster is found from the list of clusters stored in the local memory. To avoid additional calculations, the name (coordinates of the basic node that was used for cluster construction) of this cluster can be sent with a message, rather than be recalculated in each subsequent cluster. If a message have to be sent from *S* to *D*, where *S* belongs to cluster [***u***] and *D* - to cluster [***w***], then ***NC****(w)* is retrieved from the routing table, stored in the local memory for *S*, and the message is sent to cluster [***v***]=***NC****(w)*. A restriction is ***NC****(u)*×***NC****(v)*=0, so the message travel via nodes which belong either to [***u***] or to [***v***]. This procedure is implemented until ***NC****(v)*=0, which means that the next cluster has been reached. After coming to the next cluster, the process is repeated. No backtracking can happen at the routing stage.

An example of routing based on the cluster algorithm for the network of Fig.1 is given at Fig. 5. A cluster, generated by the basic node *8*, is shown. Also a path between nodes *S*=(5,3) and *D*=(3,5) is given. This path includes clusters [***5***], [***7***], [***10***] and [***8***]. Another possible path includes clusters [***3***], [***1***], [***4***], [***8***].

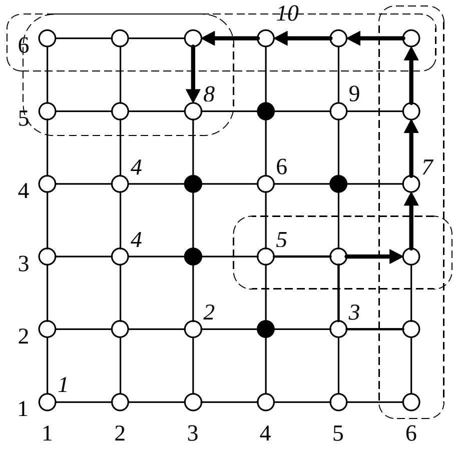

Fig. 5. Message routing using cluster algorithm

A long message can be divided to flits, according to the wormhole routing strategy. To simplify the routing procedure, each node keeps in the local memory the name of the next node for several last flits. For the each new flit it is checked, whether this information is already available.

3.5 Deadlock Elimination

Now let us consider the deadlock/lifelock problem for the proposed cluster algorithm. The problem of deadlock elimination consists of 2 parts - deadlock elimination for the routing stage and deadlock elimination for the pre-routing stage.

Since there are no faults inside a cluster, we can use any deadlock-free algorithm developed for fault-free networks, with virtual channels to eliminate deadlocks [4] for message routing inside clusters. An example is *x-y* algorithm, since for it there are no cycles in the channel dependency graph.

For pre-routing stage the number of messages that can be received by each node is restricted. It means that if we can guarantee that if the local memory is large enough, no deadlocks will occur, since stored-forward approach is in use. In fact, a local memory of the size $O(t)$ will suffice, since by Theorem 1 and Theorem 2, each node receives not more than $2(3t+3)$ at the cluster formation stage (see Section 3.1) and not more then $4(3t+1)$ messages at the cluster broadcasting stage (see Section 3.2).
To show that the developed algorithm is lifelock-free, it is sufficient to note that the length of any message path at the pre-routing and routing stages is upperbounded by the total number of nodes $N=r^2$.

Hence, the cluster algorithm is deadlock-free and lifelock-free.

3.6 Multiple Entry Nodes for Clusters

One of the important criteria for algorithm evaluation is dilation time - the difference between the length of the routing path constructed by the developed algorithm and the shortest path (in the presence of faulty nodes). As it is shown at Fig.6, it can be greater than zero for the proposed algorithm (path between node *S*=(4,1) and *D*=(2,3) is considered in this case). Shortest path contains 6 links, while for the proposed cluster algorithm this number is 7.

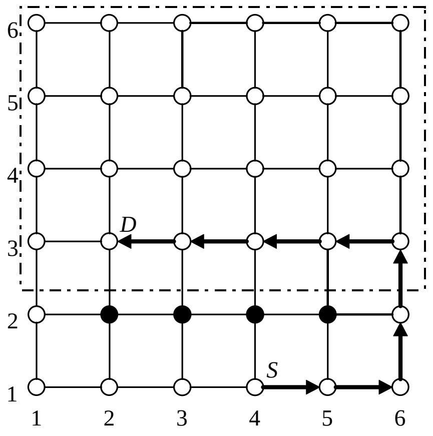

Fig. 6. Example of non-zero dilation time

To decrease dilation, not one, but several entry nodes can be stored for each cluster. For the proposed example, the destination cluster has two entry nodes - (1,3), with distance 5 from the source, and (6,3), with distance 4 from the source.

Theorem 3. If one entry point is used for every cluster, then the dilation time is equal to zero for each path containing not more than 2 clusters. If two entry points are used for each cluster, then the dilation time is equal to zero for each path containing not more than 3 clusters.

The proof of this Theorem follows immediately from the definitions of a cluster and an entry point.

With an increase in the number of entry points k, the dilation time is decreasing but the size of the routing table is increasing proportional to k.

The proposed cluster algorithm can be generalized to the case when both faulty nodes and faulty links may be present. To allow a problem-free message delivering inside a cluster, the process of cluster formation (see Section 3.1) has to be modified in this case. The first step (x-expansion) remains the same, but for y-expansion all vertical links must be checked. It results in an increase in the number of hops for cluster construction by about a factor of 2. Another change is that the procedure for constructing all neighbors of a given cluster is more complex. In this case if two clusters do not have common nodes, but have adjacent nodes, all links between such nodes must be checked. Since appropriate information must be broadcasted before construction of the cluster connectivity graph, another broadcasting stage with the same complexity as the one described in Section 3.2 is added to the developed algorithm.

If a new fault is detected, the routing tables in all nodes must be updated. Since the list of the clusters is the same in all reachable nodes, a fault-free node adjacent to a new faulty node, which will become basic, will be able to calculate the boundaries of all new clusters and broadcast this information throughout the system. We note that only clusters, containing a new fault, will be affected.

4 Algorithm Extension to the Multidimensional Case

Up to now, only 2-dimensional meshes were analyzed. The developed algorithm can be, however, extended to other network architectures, including toruses and 3-dimensional meshes.

At first, let us analyze 2-dimensional torus, where each node has 4 neighbors. In this case some clusters can be also toroid, that can be represented by setting the corresponding border values to zero. Several minor changes must be made in the procedure for cluster construction. Instead of checking for a boundary node , in this case we check whether the next node already belongs to the cluster under construction.

For a 2-dimensional torus, in is sufficient to select only two basic nodes for each fault at the cluster construction stage (for example, to the *North* and *West* from the fault).

Theorem 4. For $t>1$, a total number of clusters C for a toroidal 2-dimensional mesh is upperbounded by the min $(2t, t+\lceil r/2\rceil, \lceil r^2/2\rceil)$ and the maximal number of clusters containing the same fault-free node is upperbounded by $t+1$.

The proof of this theorem is similar to the proofs of Theorems 1 and 2.

For the 3-dimensional case, parallelepipeds will be used as clusters instead of rectangles. So, the procedure of cluster formation will be more complex- it will have three stages - expand *x*, expand *y*, expand *z* - instead of (*x*,*y*)-expansion, described in Section 3.1. Similar modifications have to be made to the algorithm, used to send messages inside a cluster. In this case the total number of clusters *C*, and number of clusters containing the same fault-free node are upperbounded by 5*t* ant 2*t*+2 correspondingly.

5 Experimental Results

In this section some results of the experimental investigation of the cluster algorithm are given. All experiments were conducted for 2-dimensional meshes.

Table 1. The results of cluster formation experiment.

r	*t*	*BN*	*C*	*IC*	*NU*
8	1	3.62	3.49	2.00	0
8	2	5.98	5.57	2.74	0.02
8	3	8.35	7.43	3.52	0.10
8	4	10.45	8.87	3.87	0.15
8	5	12.57	10.15	4.14	0.37
8	6	14.66	11.38	4.28	0.52
8	7	16.64	12.42	4.47	0.87
8	8	18.52	13.21	4.65	1.16
16	1	3.80	3.67	2.00	0
16	2	6.13	5.71	2.73	0.01
16	3	8.68	7.95	3.55	0.03
16	4	11.06	10.06	3.91	0.04
16	5	13.20	11.12	4.15	0.11
16	6	15.04	12.25	4.31	0.19
16	8	19.25	15.17	4.65	0.34
16	12	25.87	19.22	5.09	1.25

First, we analyze the process of cluster formation. Results of the experiments are given in Table 1, where *r* is the size of the mesh, *t* - the number of faulty nodes, *BN* - the number of different basic nodes, *C* - the number of different clusters, *IC* - the maximum number of clusters intersecting at one node, *NU* - the number of unreachable nodes. (All experiments are repeated 100 times and the average values are presented.)

To compare the experimental results with the theoretical estimations, in Fig. 7 the graph *C*(*t*) for *r*=8 and *r*=16 is shown. In the same graph theoretical upperbounds from Theorem 1 is given.

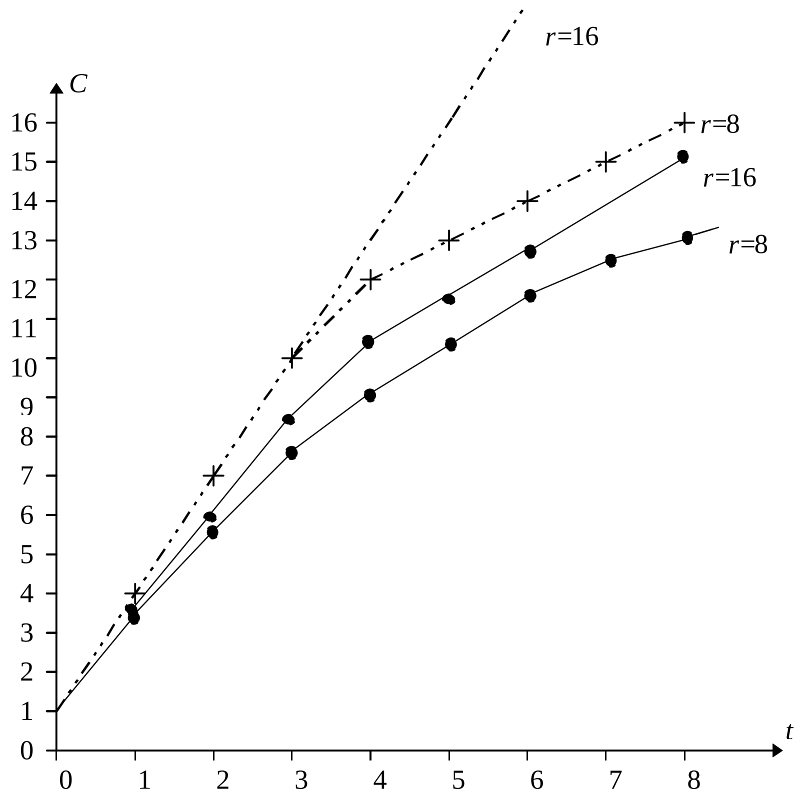

Fig. 7. Estimations on a number of clusters *C*(*t*) for *r*=8 and *r*=16 (experimental results are shown as dots).

Fig.7 shows that average numbers of clusters *C* are considerably smaller than the upperbounds given by Theorem 1 and the difference is increasing with an increase of a number of faults. Also it follows from Table 1 that almost all fault-free nodes are reachable with the proposed cluster algorithm.

In the second part of the simulation the efficiency of the resulting routing is analyzed. For randomly generated messages, the following parameters are estimated: *l* - the average length of a message path (in clusters), *L* - the average length of a message path (in nodes), L_{min} - the length of the shortest path (in nodes), $\delta = L\text{-}L_{min}$ - dilation time.

Table 2. The results of routing experiments.

r	t	l	L	L_{min}	δ
8	1	1.6	4.10	4.10	0
8	2	1.9	4.42	4.42	0
8	3	2.3	4.83	4.83	0
8	4	2.5	5.13	5.07	0.05
8	5	2.7	5.32	5.21	0.1
8	6	2.9	5.65	5.53	0.1
8	7	2.9	6.24	5.95	0.3
16	1	1.7	8.42	8.42	0
16	4	2.5	8.74	8.74	0
16	8	3.2	9.26	9.17	0.1
16	12	3.4	10.78	10.47	0.3

It follows from Table 2 that the proposed cluster algorithm provides the lengths of message path very close to the shortest ones and the dilation time grows very slowly with t.

5. Conclusions

A new adaptive algorithm for fault-tolerant deadlock-free and lifelock-free routing in two-dimensional meshes was proposed. This algorithm is local and consists of pre-routing and routing stages. The complexity of the off-line pre-routing stage is $O(tN)$, where N is the number of nodes in the system, and t is the number of faulty nodes.

The complexity of the on-line routing stage (the size of the routing table stored in the local memory) is $O(t)$. The pre-routing algorithm is performed only once, after a new fault is detected. The algorithm allows 100% of deliverable messages to be delivered in the presence of faulty nodes with no deadlocks or lifelocks. No nodes are declared unsafe. The further advantage of this algorithm is in its scaleability - it works for large numbers of faults and large numbers of nodes. The algorithm was expanded to toroid networks and 3-dimensional meshes.

6 References

1. Christopher J. Class and Lionel M.Ni, The Turn Model for Adaptive Routing. / *Proc. of the 19th Annual Int. Symp. on Computer Architecture*, pp. 278-286, May 1992.
2. C. Cunningham and D. Avresky, Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes. / *Proc. of First Int. Symp. on High Performance Computing Architecture*, Raleigh, North Carolina, USA, January 1995.
3. C. Cunningham and D. Avresky, Fault-Tolerant Adaptive Broadcasting and Multicasting using Wormhole Routing in Two-Dimensional Meshes. / *Technical Report 95-033*, Department of Computer Science, Texas A&M University.

4. J. Duato, A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. / *Proc. of Int. Conf. on Parallel Processing*, vol. I., pp. 142-149, August 1994.
5. R.V. Boppana and S. Chalasani, A Comparison of Adaptive Wormhole Routing Algorithms. / *Computer Architecture News*, 21(2), pp. 351-360, May 1993.
6. S. Chalasani and R V. Boppana, Communication in Multicomputers with Nonconvex Faults. / *IEEE Trans. on Computers*, vol. 46, pp. 616-622, May 1997.
7. H.-L. Chen and N.-F. Tzeng, Subcube determination in faulty hypercubes. / *IEEE Trans. on Comput.*, vole.46, pp. 871-879, August 1997.
8. L. M. Ni and P.K. McKinley, A Survey of Wormhole Routing Techniques in Directed Networks. / *Computer*, vol. 26, pp. 62-76, February 1993.
9. W. Dally and C.L. Seitz, Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. / *IEEE Trans. on Comput.*, vol. 36, pp.547-553, May 1987.
10. Y.M. Boura and C.R. Das, Fault-Tolerant Routing in Mesh Networks. / *Proc. of Int. Conf. on Parallel Processing*, vol. O., pp. 106-109, August 1995.
11. R.V. Boppana and S. Chalasani, Fault-Tolerant Wormhole Routing Algorithms in Mesh Networks. / *IEEE Trans. on Comput.*, vol. 44, pp.848-864, July 1995.
12. W.J. Dally and H. Aoki, Deadlock-Free Adaptive Routing in Multiprocessor Networks Using Virtual Channels. / *IEEE Trans. on Parallel and Distibuted Systems*, vol. 44, pp. 466-475, April 1997.