IEE Proc. Computers and Digital TechniquesVol. 147, No.6, November 2000, pp. 397-402

Fault-tolerant Wormhole Routing using a Variation of Distributed Recovery Block Approach

Gul N. Khan
Department of Electrical Engineering
University of Saskatchewan, 57 Campus Drive
Saskatoon SK S7N 5A9 Canada

Gu Wei
Biztone.Com Pte Ltd.
65 Club St,
Singapore 069439

Abstract: This paper describes a fault-tolerant wormhole routing technique that incorporates a variation of distributed recovery block (DRB) approach. The section of a parallel system that spans between the source and destination nodes is dynamically partitioned into overlapping DRB groups. A DRB group consists of a current node, a primary and an alternate successor node. The message packets travel towards the destination from one DRB group to the next group. A prototype of the routing system is implemented for mesh and hypercube topologies; however, the method can be used for topologies with a minimum node connectivity of three. The simulation results indicate that the DRB approach based wormhole routing tolerates both node and link failures.

1 Introduction

Inter-processor communication is an important and essential activity for parallel and distributed processing. In distributed memory parallel systems, each node is directly connected to a small fraction of other nodes. A message passing system provides a virtual connectivity by representing a fully connected view of the system. Message propagation, path establishment and deadlock avoidance are the main strategies incorporated in message passing systems. There are three types of message propagation techniques namely store-and-forward, wormhole and virtual cut through. Wormhole routing is commonly used in the recent generation multi-computers due to lower latency and small buffer requirement.

In wormhole routing, a packet is transmitted as a contiguous stream of flits that occupy a sequence of nodes and communication channels along the path. After receiving a header flit, an intermediate node determines the next node to forward the message. The header flit determines the path and the tail flit releases it as the packet travel towards its destination. If the header is blocked due to the non-availability of a communication channel or a network component failure, the remaining flits of the message get stranded in the network. Adaptive wormhole routing techniques have been introduced to handle the problem of message blocking. Gaughan and Yalamanchili have used a pipelined circuit-switching mechanism with backtracking for fault-tolerant routing systems [5]. Virtual channels have been employed for adaptive routing [4]. A number of adaptive and fault-tolerant wormhole routing techniques have been developed for mesh topologies [2, 3, 6, 11 and 12], that have been reviewed recently [13]. We present a new fault-tolerant wormhole routing technique that is based on distributed recovery block approach [9]. The method can route messages for various network topologies with a node connectivity of three or more.

1.1 Preliminaries and Assumptions

A Q_n hypercube contains 2^n nodes and a relatively small diameter with $n2^{n-1}$ links. The degree of each node is n and every node has a distinct *n-bit* binary address. The addresses of two neighboring nodes differ by exactly one bit. The number of bits by which any two nodes x and y differ is defined as:

$$D(x, y) = \Sigma d(x_i, y_i)$$
$$\text{where} \quad d(x_i, y_i) = 1, \text{ if } x_i \neq y_i$$
$$d(x_i, y_i) = 0, \text{ if } x_i = y_i \qquad (1 \leq i \leq n)$$

If a message, *msg* is to be routed from a source *x* to a destination node *y*, the message format can be defined as: $(D(x, y), [c_1, c_2, \ldots, c_k], msg)$

where $k = D(x, y)$ is the Hamming distance between the nodes *x* and *y*.
$[c_1, c_2, \ldots, c_k]$ is the coordinate sequence of different bits of node addresses.

For notational purposes, each link is presented by a binary string with a "-" symbol at the corresponding index. For instance a link between 0000 and 0100 nodes is represented by link 0-00. Following assumptions are made to present the fault-tolerant wormhole routing technique.

- Source and destination nodes are healthy and fault-free.
- A node with no healthy links incident on it is considered faulty.
- Faults are non-malicious and a failed component simply ceases to work.
- The deadlock avoidance is not explicitly considered in our implementation.

2 DRB Approach and Fault-tolerant Message Routing

Recovery block (RB) is a basic software fault tolerant technique that can tolerate both hardware and software faults. The completion time of a recovery block improves by its distributed implementation [9]. In the distributed recovery block (DRB) approach, the primary and alternate versions of a software module are executed concurrently on different nodes of a parallel system. We have adapted the DRB scheme to route messages in networks with a node connectivity of three or more [8]. The section of the parallel system that spans between the source and destination nodes is partitioned into overlapping DRB groups. The formation of DRB groups in a mesh is explained in Fig. 1 for message routing from node 11 to 44.

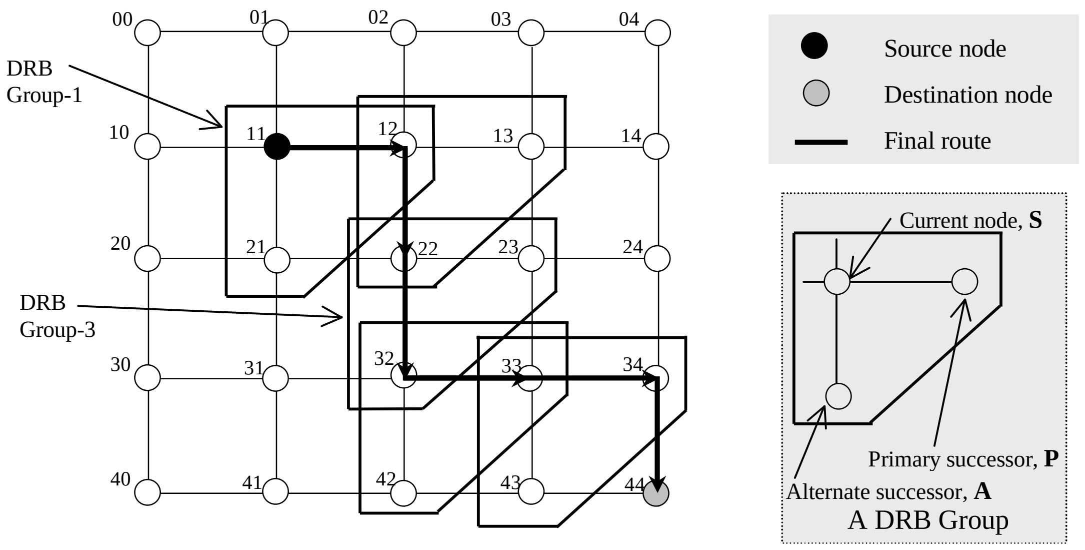

Fig. 1 Formation of DRB Groups

A DRB group consists of a current node, primary and alternate successor nodes. The current node (S) delivers the message to both successors. The primary successor node (P) and alternate successor (A) each has a set of try and acceptance test as shown in Fig. 2. After receiving the message, both successor nodes apply the acceptance test (AT). The time acceptance test is used to ensure a timely message delivery. In a fault-free situation, primary and alternate successors pass the AT. The primary node notifies its success to alternate successor and forwards the message to next DRB group. If the primary successor node or its communicating link fails while alternate passes the AT, their role exchange. Different failure scenarios are explained in Fig 3. When both successors fail and can not recover, other successor nodes are selected. In a 2D mesh, there is only one more successor. For hypercube, f_2(*current*, *destination*) and t_2(*current*, *destination*) is chosen as primary and alternate successors respectively. The $\boldsymbol{f_i}(x, y)$ is the position of the *i*th bit from the left, i.e. the dimension *i*, in which *x* and *y* nodes differ [10]. We introduce a similar function $\boldsymbol{t_i}(x, y)$ that determines the position of an *i*th difference bit between *x* and *y* from the right. For example:

$f_1(001101, 001010) = 4 \qquad f_2(001101, 001010) = 5$

$t_1(001101, 001010) = 6 \qquad t_2(001101, 001010) = 5$

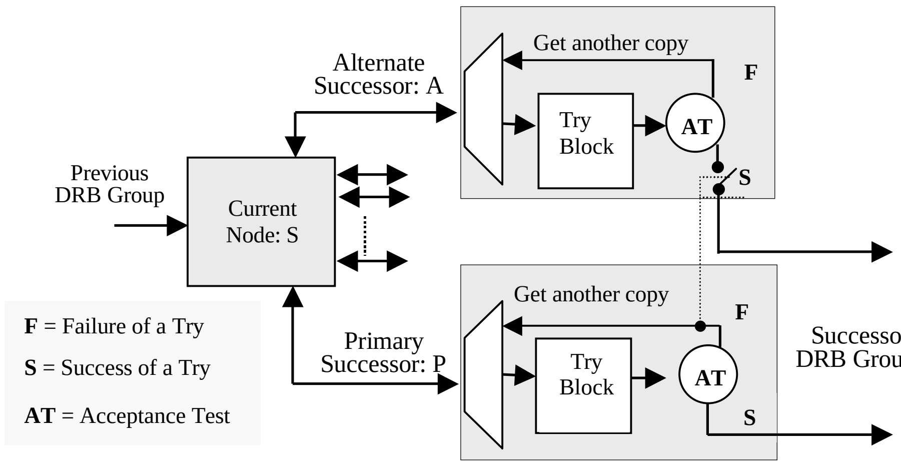

Fig. 2 A DRB Group for Fault-tolerant Message Routing

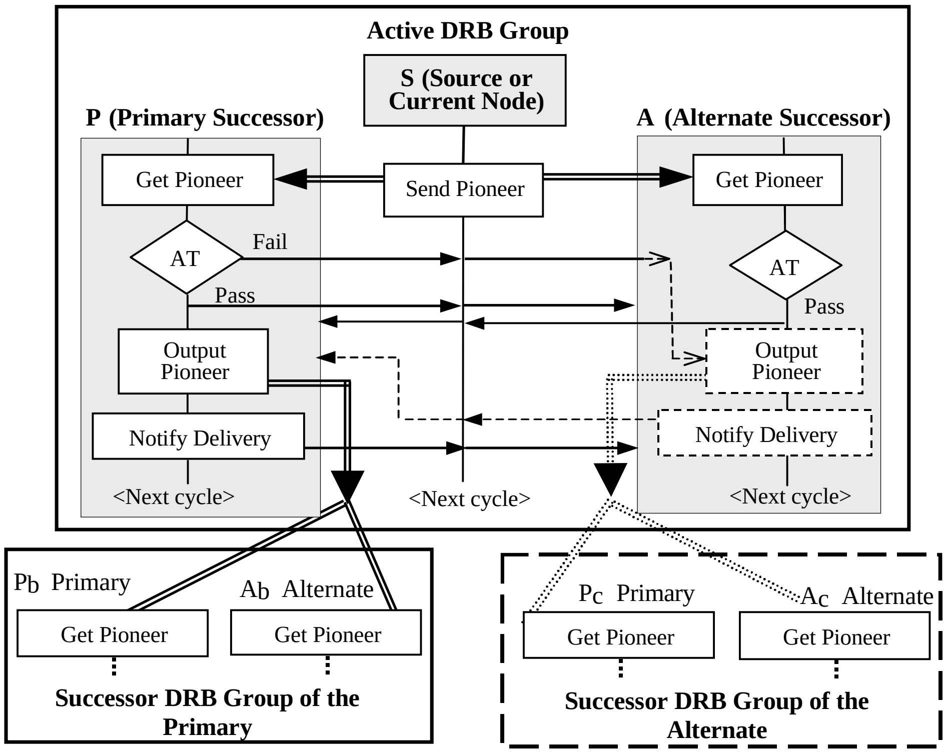

Fig. 3 Fault-Free Operation and Fault Recovery from Primary Failure

3 Fault-tolerant Wormhole Routing

In a distributed system, there are different ways of organising the fault information needed by routing algorithms. It can be either made available globally or at the local level. The global fault model based routing algorithms must have alternate means of transmitting fault information during the transition stage of fault detection. In our fault-tolerant routing technique, nodes use local fault information. The message routing process has two phases: path establishment and message transfer. Fault-free path establishment is a critical process and we employ DRB approach to find a fault-free path.

3.1 Fault-free Path Establishment

A pioneer flit is routed by using the DRB approach to establish a fault-free and economical path. In contrast to the backtracking protocol [5], packet data flits can follow the pioneer flit immediately to overlap message transfer and path establishment processes. The source node sends the pioneer flit to its two successors as explained in Fig. 3. The successors acknowledge and if the acknowledgement is received from both successors, the primary successor has the priority to act as current node in the next DRB group. Otherwise, the successor that acknowledges will become the next current node. The next current node sends the pioneer flit to its successors and the process is repeated until the destination is reached.

Consider a Q_4 hypercube shown in Fig. 4a, whose links 11-0 and 01-0 are faulty and a message is to be routed from node, 1111 to a destination node 0000. To find a minimal fault-free path, the source node 1111 sends the pioneer flit to its primary and alternate successor nodes, 1110 and 0111. When there is no fault, the node 1110 acts as a current node and forwards the pioneer to its primary and alternate successors (1100 and 0110). The primary successor does not receive the pioneer flit due to a faulty link, 11-0 and the alternate successor takes over. It sends the pioneer to its successor nodes, 0100 and 0010 as illustrated in Fig. 4b. The primary successor, 0100 fails due to another faulty link 01-0 and alternate node forwards the pioneer flit to the destination. A fault-free path is established in the form of 1111→1110→0110→0010→0000. Pseudo code for the distributed implementation of path establishing algorithm is provided in Fig. 5.

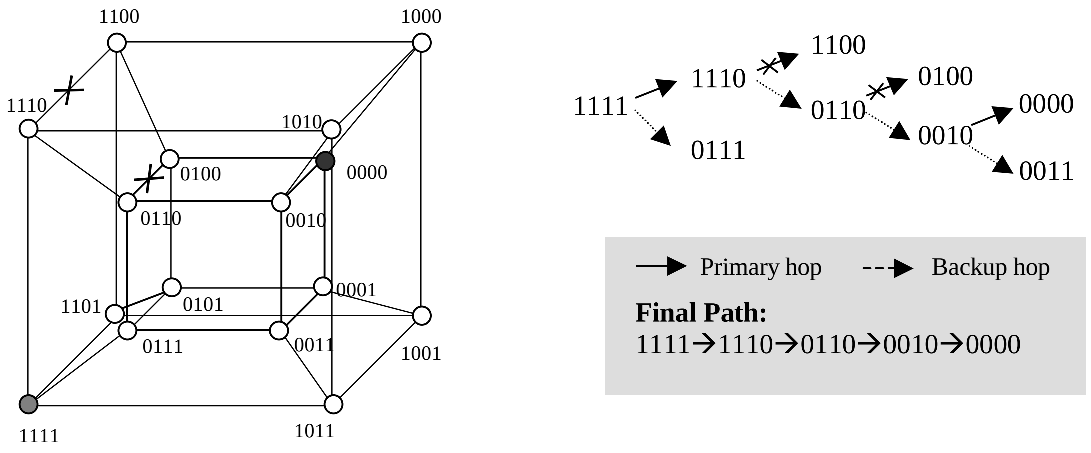

Fig. 4 (a) An Injured Q_4 Hypercube (b) Establishing a Fault-free Path

```
Current node: curr
Begin  for every pioneer
  if curr == dest then the pioneer has arrived the destination;
   else begin
    pri_pass = alt_pass = FALSE;
    while [not (pri_pass) & not (alt_pass)] or [all tries exhausted]
      begin
         find pri and alt among the neighbor nodes;
          /* For hypercube use f_i(x,y) and t_i(x,y) functions */
         send (pioneer) to pri and alt;
         if curr ≠ src
            then send (conf) to partner; /* via pre_curr */
         receive (pri_pass) from pri with time out-1;
          /* in the case of time out-1, pri_pass = FALSE */
         receive (alt_pass) from alt with time out-2;
          /* time out-2 < time out-1 */
          /* for time out-2:alt_pass = FALSE */
         send (pri_pass) to alt;
         send (alt_pass) to pri;
      end;   /* while */
      if not (pri_pass) then send (pioneer) to pri;
      if not (alt_pass) then send (pioneer) to alt;
      pre_curr = curr;
   end;   /* else begin */
end.   /* main curr */
```

(a) Current Node

```
Primary Successor node: pri
begin
   pri_pass = alt_pass = FALSE;
   while [not (pri_pass) & not (alt_pass)] or [all tries exhausted]
   begin
      receive (pioneer) from curr;
      execute AT;
      pri_pass = AT result;
      send (pri_pass) to curr;
      receive (alt _pass) from curr with time-out;
         /* in the case of time-out, alt_pass = FALSE */
   end;   /* while */
   if pri_pass  then curr = pri;
   else begin
      receive (pioneer) from curr; execute AT; partner = pri;
      if (curr ≠ src) begin
         while not (time-out)
            receive (conf) from pre_curr;
            if not (conf) then curr = partner;
      end;   /* if begin */
   end;   /* else begin */
   end.   /* main pri */
```

(b) Primary Successor Node

Notations: *src* = source node; *dest* = destination node; *curr* = current node; node.
alt = alternate successor node; *pri* = primary successor; *pioneer* = pioneer or header flit
conf = confirmation of a successful delivery of the pioneer flit.
pri_pass = AT result from primary successor; *alt_pass* = AT result from alternate successor.
pre_curr = current node of the previous DRB group.
partner = partner node of *curr* in the previous DRB group.

```
Alternate Successor node: alt
begin
   pri_pass = alt_pass = FALSE;
   while [not (pri_pass) & not (alt_pass)] or [all tries exhausted]
   begin
      receive (pioneer) from curr; execute AT;
      alt_pass = AT result; send (alt_pass) to curr;
      receive (pri_pass) from curr with time-out;
          /* in the case of time-out, pri_pass = FALSE */
   end;   /* while */
   if (alt_pass) & not (pri_pass) then curr = back;
   else begin
      if not (alt_pass) begin
         receive (pioneer) from curr; execute AT;
      end;   /* if */
      partner = alt;
      if (curr ≠ src) begin
         while not (time-out)
            receive (conf) from pre_curr;
         if not (conf)  then curr = partner;
      end;   /* if begin */
   end;   /* else begin */
   end.   /* main alt */
```

(c) Alternate Successor Node

Notations: *src* = source node; *dest* = destination node; *curr* = current node; node.
alt = alternate successor node; *pri* = primary successor; *pioneer* = pioneer or header flit
conf = confirmation of a successful delivery of the pioneer flit.
pri_pass = AT result from primary successor; *alt_pass* = AT result from alternate successor.
pre_curr = current node of the previous DRB group.
partner = partner node of *curr* in the previous DRB group.

Fig. 5 Fault-free Path Establishment Algorithm Pseudo Code
(a) Current Node **(b)** Primary Successor Node **(c)** Alternate Successor Node

3.2 Message Transfer

Two possibilities that can happen during a message transfer are:

- No faults occur in the established fault-free path.
- Communication links and/or nodes on the fault-free path fail.

The first is an ideal situation and the message reaches the destination without any hurdle. The possibility of a relevant network-component failure during a message transfer is realistic. If it happens, the nearest healthy node from the destination where the packet flit is blocked can act as a temporary source node. It establishes another path to the destination by using the same DRB approach. The pioneer flit information is kept at each intermediate node on the routing path until the message tail is passed. After establishing a new path, the blocked portion of the message is forwarded. If the message is blocked again due to another fault, the same procedure is repeated. A message may be split into two or more sub-messages that are routed on different paths and all the sub-message are assembled at the destination.

Consider once again the injured hypercube Q_4 of Fig. 4a where a message is routed from node 1111 to 0000. Assume that when some portion of the message has passed the node 0110, another communication link 0-10 fails. The intermediate node, 0110 serves as a source node and finds another available path to the destination. It sends the pioneer to its successor nodes, 0100 and 0111 excluding the faulty link 0-10. The primary successor fails due to a faulty link 01-0 and the alternate successor, 0111 takes over and sends the pioneer to its successors. If there is no further failure, message is forwarded via 0110→0111→0101→0100→0000. The first part of the message has been routed via 1111→1110→0110→0010→0000.

There are two options to deal with the failures during message transfer.

- *Buffer the Blocked Message at the Intermediate Node*
  When a communication link or node on the fault-free path fails during a message transfer, the blocked part of the message can be buffered at the healthy intermediate node like a virtual cut through [7]. Buffering the blocked message spares the network resources, however, intermediate nodes require additional buffering space and latency of the blocked message will increase.
- *Blocked Message is Kept Stranded in the Network*
  The blocked message is not buffered at the intermediate node. The node, where the message is blocked, establishes a fault-free path and routes the message to the destination. This option requires very little storage; however, other network traffic suffers.

4 Prototype Implementation and Experimental Results

A prototype of the fault-tolerant routing technique is implemented on a nine-node IBM SP parallel system. The system is configured to realize mesh and hypercube topologies. The system nodes are fully connected by a high performance switch, however, their interconnection is restricted to configure mesh, hypercube, and other topologies. Any direct communication between the non-neighbouring nodes of a topology is barred. MPI is the basic communicaiton software used in the simulations. The flit size used for wormhole routing is four bytes as it is the minimum data unit supported by the MPI implementation.

The routing system is implemented for a 3x3 2D-mesh shown in Fig. 6. Message latency is obtained by routing different size messages for various fault scenarios. Results from some of these faults are presented for routing messages from node 22 to 00.

- Link Failure-a: Communication links fail before the routing start (Fig. 6a).
- Link Failure-b: In addition to Link Failure-a, another link fails during message transfer (Fig. 6b).
- Node Failure: Node failure before the routing starts (Fig. 6c).

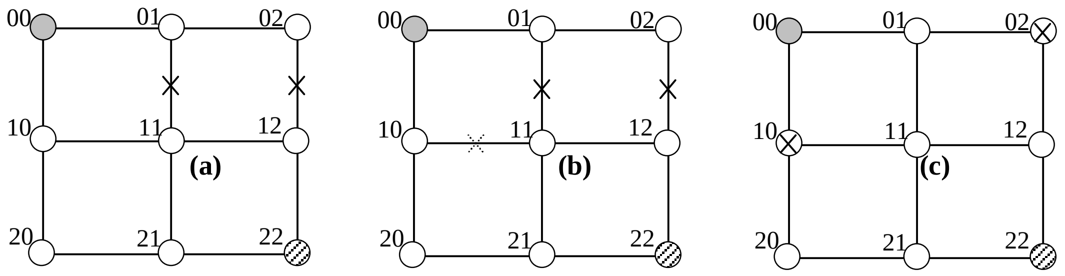

Fig. 6 Failure Scenarios for 2D Mesh
(a) Link Failure-a (b) Link Failure-b (c) Node Failure

In the first case, links between 01→11 and 02→12 are faulty before the start of routing. The initial fault-free path established by the pioneer flit is 22→12→11→10→00. For Link Failure-b, an additional link, 10→11 fails during message transfer and a portion of the message gets blocked at node 11. The intermediate node (11) serves as a source node. It buffers the blocked message portion and finds a new fault-free path: 11→21→20→10→00. The message latency for these cases is plotted in Fig. 7, which indicates a higher latency for Link Failure-b. It happens due to buffering the blocked message, establishing a new and longer routing path.

A similar simulation is conducted for an injured Q_3 hypercube of Fig. 8 having different faulty components. The messages are routed from node 111 to 000 and some failure cases are presented here.

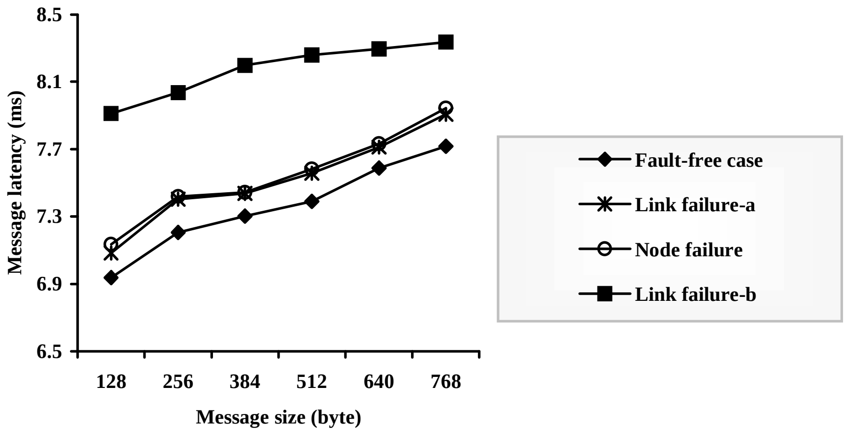

Fig. 7 Fault-tolerant Wormhole Routing for a 2D Mesh

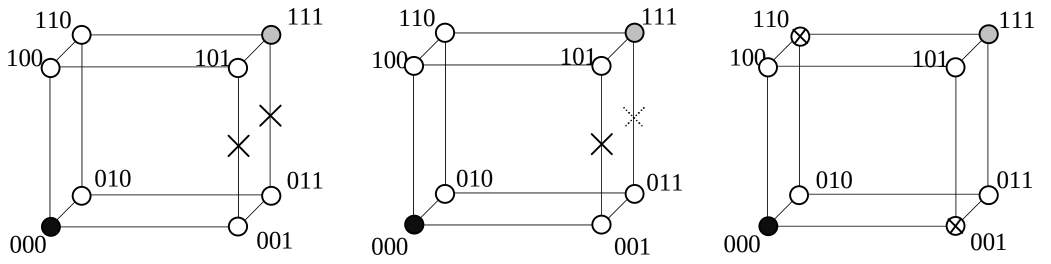

Fig. 8 Failure Scenarios for Q3 Hypercube:
(a) Link Failure-a (b) Link Failure-b (c) Node Failure

The Link Failure-a (Fig 8a) represents communication link (-01 and –11) failure before message routing. A fault-free path (000→001→011→010→110→111) is found around these faulty links. In the case of Link Failure-b (Fig 8b), a communication link, -01 is faulty at the beginning and a fault-free path (000→001→011→111) is determined. Afterward, another communication link, -11 fails during the message transfer. The part of the message gets blocked at node 011. This node acts as a temporary source node and determines another available path (011→010→110→110). The blocked message flits are not buffered and they are routed to the destination via the new path. The path is identical to Link Failure-a path but latency is little bit larger as shown in Fig. 9. Investigation is also conducted for node failures shown in Fig 8c. Latency is plotted in Fig. 9 for all the failures and the results indicate that DRB approach based method can route messages for node and link failures.

Additional experiments are conducted to evaluate dynamic failures happening during message transfer. Consider the injured Q_3 hypercube of Fig. 8b again. A message is to be routed from node 000 to 111 and link -11 fails during the message transfer. A portion of the message gets blocked at node 011 that can be either buffered or kept stranded in the network. In the first simulation, it is assumed that there is no traffic in the network. The message latencies for both the options are plotted in Fig. 10. It is evident from the results that the latency improves when the message is kept in the network.

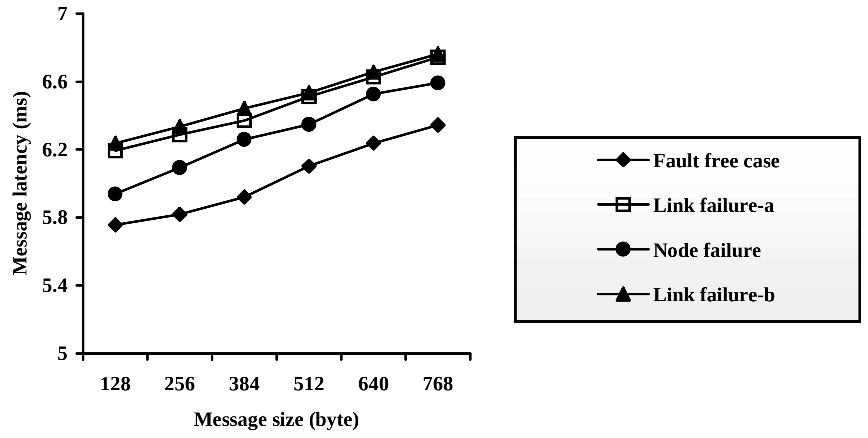

Fig. 9 Fault-tolerant Wormhole Routing for a Q_3 Hypercube

However, when the message is not buffered, it occupies network resources. Additional simulations are conducted to analyze its effect. The network is flooded with messages by generating messages continuously from each node to random destinations. Average network latency is determined from the total network traffic latency. We contrast the network latencies for both options and the results are presented in Fig. 11. It is observed that for message buffering, the network latency is not affected considerably. On the contrary, for a purely wormhole routing, there is a sizable increase in latency of the network traffic.

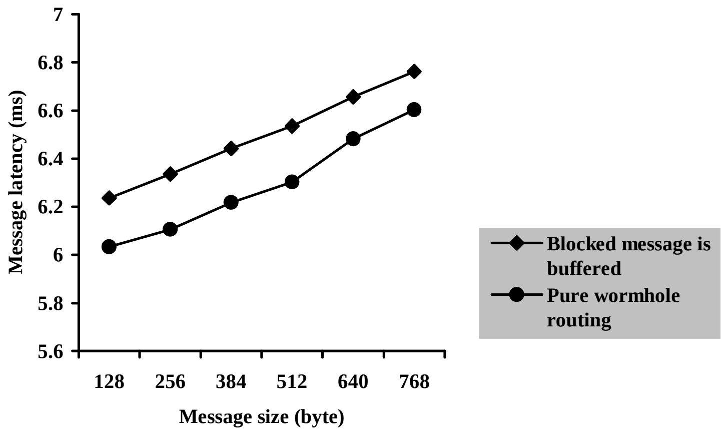

Fig. 10 Message Latency when the Message is either Blocked or Buffered

5 Deadlock Avoidance and Recovery

In wormhole routing, once a communication link or channel accepts the header flit of a packet, rest of the packet must be accepted before accepting any other packets. A cyclic wait condition causes deadlock when a packet holds channels while waiting and excludes other packets from acquiring the held channels. One can virtualized the network by introducing virtual channels and defines routing functions for the virtual networks with no cycles. A number of deadlock avoidance schemes for adaptive routing in k-ary n-cubes have been proposed and implemented by using virtual channels [3, 4, 11 and 12]. A similar technique can be adapted for the DRB based fault-tolerant routing. Virtual channels need to be introduced at the DRB group level and within a group for deadlock free adaptive routing. The virtual channel management can be simplified by partitioning them hierarchically in terms of inter-node and inter-group channels.

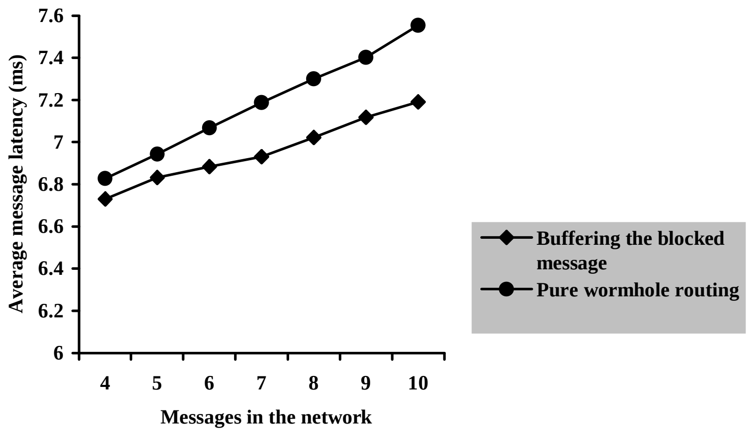

Fig. 11 Impact of Message Blocking on Network Traffic Latency

DRB based routing technique presented in this paper is fault-tolerant and it recovers from deadlocks if one considers the deadlock as a failure. When a message is blocked due to a deadlock and can not move towards its destination, the successor doesn't acknowledge the header delivery. The current node that holds the header-flit treats the deadlock as a failure and tries the alternate successor. It can even select additional successors as described by the algorithm stated in Fig. 5. Similar failures during the message transfer are simulated for the prototype implementation. The network is flooded with a continuous stream of messages from each node. No deadlocks are observed for these simulations being executed for more than an hour. Deadlock recovery is an excellent alternative to deadlock avoidance, specifically for fault-tolerant routing algorithms. We have employed two types of time-out intervals to detect network component failure. These time-out intervals can be tuned to detect deadlocks effectively. In this way, the DRB based routing algorithm can recover from deadlocks easily. A similar approach has been used in *disha* [1]. In the rare case when a deadlock message can not be routed at all, the current node absorbs the blocked message and later re-injects it for routing it to the destination. The method presented in this paper is so flexible that it can even absorbs the part of a packet and then routes it to the destination.

6 Conclusions

A new fault-tolerant wormhole routing technique is presented, which is based on the DRB approach. The routing process consists of fault-free path establishment and message transfer. Some of the existing techniques including the backtracking determine the fault-free path only and assume that no more faults will occur during the message transfer [5]. Some previous fault-tolerant wormhole routing techniques only consider a limited number of node or link failures for evaluation [13] while other deals with only mesh topologies [2, 6, 11 and 12]. The DRB based routing suits to all regular and irregular topologies with a minimum node connectivity of three. It also caters for nodes or links failures and the failures can be static or dynamic. The prototype wormhole routing system routes messages successfully from one node to another, if a single healthy path exists between them. The results from the investigation show that the DRB approach based wormhole routing is an effective way of assuring the message delivery in faulty networks. It is not necessary for the nodes to know the status of its neighboring nodes. The DRB based routing has also been compared with backtracking based fault-tolerant message routing. The results indicate that DRB approach based wormhole routing has lower latency and higher throughput [14].

7 References

1 ANJAN, K. V., and PINKSTON, T. M.” ‘An efficient, fully adaptive deadlock recovery scheme, DISHA’, Proceedings International Symp. on Computer Architecture, June 1995, pp. 201-210

2 BOPPANA, R. V., and CHALASANI, S.: ‘Fault-tolerant wormhole routing algorithms for mesh networks’, *IEEE Trans. Computers*, 1995, **44** (7), pp. 848-864

3 CHIEN, A. A., and KIM, J. H.: ‘Planar-adaptive routing: low-cost adaptive networks for multiprocessors’, *Journal of ACM*, 1995, **42** (1), pp. 91-123

4 DALLY, W. J., and AOKI, H.: ‘Deadlock-free adaptive routing in multicomputer networks using virtual channel’, *IEEE Trans. Parallel and Distributed Systems*, 1993, **4** (4), pp. 466-475

5 GAUGHAN, P. T., and YALAMANCHILI, S.: ‘Adaptive routing protocols for hypercube interconnection networks’, *IEEE Computer*, 1993, **26** (5), pp. 12-23

6 GLASS, C. J., and NI, L. M.: ‘Fault-tolerant wormhole routing in meshes without virtual channels’, *IEEE Trans. Parallel and Distributed Systems*, 1996, **7** (6), pp. 620-636

7 KERMANI, P., and KLEINROCK, L.: ‘Virtual cut-through: a new communication switching technique’, *Journal of Computer Networks*, 1979, **3** (11), pp. 267-286

8 KHAN, G. N., and WEI, G.: ‘Adaptive and fault-tolerant message routing using distributed recovery block approach’, *Proceedings Parallel and Distributed Computing and Systems (PDCS ’99)*, Cambridge, Massachusetts, USA, 3-6 November 1999, pp. 732-737

9 KIM, K. H., and WELCH, H. O.: ‘Distributed execution of recovery blocks: an approach for uniform treatment of hardware and software faults in real-time applications’, *IEEE Trans. Computers*, 1989, **38** (5), pp. 626-636

10 LEE, T. C., and HAYES, J. P.: ‘A fault-tolerant communication scheme for hypercube computers’, *IEEE Trans. Computers*, 1992, **41** (10) pp. 1242-1256

11 LIBESKIND-HADAS, R., and BRANDT, E.: ‘Origin-based fault-tolerant routing in the mesh’, *Future Generation Computer Systems*, 1995, **11** (6) pp. 603-615

12 LINDER, D. H. and Harden, J. C.: ‘An adaptive and fault-tolerant wormhole routing strategy for k-ary n-cubes’, *IEEE Trans. Computers*, 1991, **40** (1) pp. 2-12

13 MOHAPATRA, P.: ‘Wormhole routing techniques for directly connected multicomputer systems’, *ACM Computing Surveys*, 1998, **30** (3), pp. 374-410

14 WEI, G.: ‘Fault-tolerant message passing system’. M.Sc. Thesis, 2000, Nanyang Technological University, Singapore