Fault Tolerant Routing in Mesh Topologies

Anshuman Tripathi
07CS3024
anshuman.iit@gmail.com

Manaal Faruqui
07CS3011
me@manaal.com

1 Introduction

Fault tolerant routing refers to transfer of messages from a source to destination in a network with faulty links and/or nodes. Fault tolerant routing is one of the key problems in designing robust distributed algorithms. Internet routing protocols like BCG are also examples of adaptive routing algorithms.

Adaptive routing in distributed systems can be desirable at either the application level (like routing in LAN) or at the harware chip level, where on chip in the circuit layout wants to communicate to another chip (for example two processors communicating intermediate results). In both these cases the main aim of the underlying routing problem differs, for example in the case of adaptive routing in LAN it is more desirable to optimize upon the number of hops or the throuhput of the network giving lesser weightage to the initial routing latency. However when it is the case of on-chip routing the optimization parameter is more likely to be the time required to route the message, the message over-head, complexity of the algorithm (as it has to be implemented at hardware level). In the case of on-chip routing, the network topology, in general, is not arbitrary but a well defined layout pattern. This enables optimization of an algorithm for the specific topology.

Mesh topology is a very common circuit layout where the chips/nodes are arranged in a $N \times N$ matrix with each node connected to its immediate column and row neighbors. In this work we shall discuss and compare adaptive routing algorithms for mesh topologies to guarantee routing along the shortest path.

2 Literature Review

Fault tolerance is achieved in a routing algorithms either from redundant message copies in the network or redundant message routes. Many algorithms induce like *flooding* (resembles a distributed BFS on the interconnection network), *gossip* (flood selective message copies based on information from neighbours), *stochastic communication* [2] and *N-random walk* [3], induce multiple message copies in the network. Although these routing algorithms generally deliver shortest route and good tolerance against faults, they are extensive in terms of power consumption power consumption and showcase low resiliency for congestion.

Other widely used algorithms, like *dimension order routing* OR *xy-routing* [8], exploit presence of redundant routes in the network and can be bifurcated in those that require global knowledge of the network state and those that are distributed in control. The amount of global knowledge required by the algorithms dictates the implementation complexity, scalability, amount of pre-processing and message complexity. These algorithms involve simple decisions by every node, thus are faster, cheaper to implement and economical in terms of power consumption. However they don't guarantee a shortest path route to the destination.

Algorithms that use global knowledge can provide better routes to the destination but on the cost of additional implementation complexity (because of the hardware required to make the global state available to all the nodes). The additional hardware consumes more chip area, resulting in coarser design, and more power. Some routing protocols that use global knowledge or partial global knowledge include *distance vector routing*, *link state routing* [15], *DyNoC* [1] and protcols based on *DSPIN* [18].

Deadlock is on of the most common problems that may appear in a distributed algorithm, and with respect to the routing algorithms when there is such a cyclic dependency of packets that one node waits for message from other node and none of the nodes can proceed then the system is in a state of deadlock (as opposed to livelock where the message is forwarded for indefinetly many hops without delivery). The problem of deadlock in a routing algorithm can be overcome in by using one of the two strategies:

1. *Deadlock avoidance* : Some well known avoidance methods include *turn model* [6], [9] and *virtual channels* [8].

2. *Deadlock recovery*: A few examples of recovery are illustrated in [8] and [17].

Routing Algorithms that are completely distributed in thier control have been presented in [6] and [9] where *odd-even turn model* has been used as a deadlock avoidance scheme and the protocols have been analysed on the metrics of resilience to network congession.In [13] a routing algorithm is presented, which routes the packet into progressive x or y-direction, whichever has the smaller amount of other traffic. The method is called dynamic xy routing. A similar approach is presented in [10], where also non-progressive routes are allowed. The paper claims to achive 1.95 times reduction in latency and 3.15 time reduction in energy consumption over the trivial directed flooding method. A deadlock free distributed routing algorithm has been proposed in [12] for routing messages in an on-chip interconnection network while presenting a thorough comparative analysis with some of the previously mentioned protocols like *xy-routing* [8], *N-random walk* [3] and others.

In [4] a distributed algorithm has been proposed for handeling the adaptive routing problem in mesh topologies with only non-convex faults more efficiently while using only three virtual channels for ensuring deadlock freeness. Similar is the case with *Origin-based routing* [14] protocol for mesh topologies, which is a partial distributed algorithm, where the delivery of message is ensured with any back tracking given some prior knowledge (of bounding boxes of the faults). An algorithm *FT-Ecube* has been proposed in claiming to use only two virtual channels and also tolerates *f-chains* in meshes. The works also illustrates that *FT-Ecube* is deadlock-free and livelock-free, in meshes when it has non-overlapping multiple *f-regions*, while also giving good performace in terms of average latency. A fully adaptive routing algorithm *FTRoute* has been suggested in [16] for square and hexagonal meshes, and the work claims to achive a shortest path routes with very high probabilities. The algorithm proposed is a simple modified version of a greedy path finding algorithm (with some modifications to prevent cycles and ensure delivery if path exits).

A considerable amount of work has also been done for fault tolerant routing in *hyper-cube* architecture. Author of [5] explores an algorithm that does not require any global knowledge of the faults but uses hyper-cube specific properties because of which it can tolerate n faults for an n-dimension hyper-cube, but since it uses specific properties to hyper-cube it is not applicable to mesh architectures. Similarly [7] and [11] present apdaptive routing algorithms for hyper-cube architectures but they are either not applicable to mesh topologies or use some sort of global information.

Thus in literature many approaches have been proposed for solving the problem of adaptive routing in interconnection networks. These approaches are different basically because of the routing parameters they try to optimize (eg. routing delay, message overhead) or the topologies for which they are applicable or optimized (eg. mesh topology [14], hyper-cube [5]).

3 Problem Statement

The Problem of fault tolerant routing is quite general and diverse, we shall contain ourselves to the prrblem of fault tolerant routing in mesh topologies. To the best of our knowledge there has been no deadlock free fully adaptive routing algorithm for the mesh topologies that ensures that the message is routed along the shortest path. There have been some proposed algorithms [16] that follow the shortest route with very high probabilities. Shortest path routing is important because once the shortest path has been established the routing may take less energy and less latency.

In the present work we propose an adaptive algorithm for routing along the shortest path in mesh topologies with out any global knowledge of the faults. We assume that a nodes knows the status of all its neighbours (i.e faulty or not). It is also assumed that a node can declare itself faulty (by ceasing to respond to `keep-alive` messages). The faults can be either convex or concave in nature. It is assumed that each message has a default time to live (ttl) value after which it expires (i.e. is discarded from the network).

4 Proposed Solution

In the proposed algorithm we assume that each message has a unique ID `msgid`. Without the loss of generality we can assume that the faults are convex in shape because as we shall see later in that concave faults can be easily converted to convex faults and furthermore the concave faults does n't affect the correctness of the algorithm, it just makes the number of back tracks more (in the stabalization phase). Each node has 4 links (other than the border nodes which have 3 links). The routing algorithm proposed herein is equivalent to the distance routing protocol used in LAN comminications. Therefore just like the RIP protocol there is a stabilization time for the system. Within the granuality if that stabilization time the route taken by the protocol will be shortest path route. Though the routing algorithm proposed herein doesn't require routing table, it can still be used as a cache for faster computations of the shortest path. The routing table has the following entries:-

`destination` : This entry is the destination of the message to be sent or forwarded, if the `destination` of an outgoing message maches the value in this field then the message is forwarded along the link specified in `nexthop`. The default rule for routing is denoted by the value "*" in this field. Thus the value "*" will match any destination in the messsage `destination` field, for this reason the entry "*" is always kept at the end of the routing table.

`nexthop` : This field in the routing table denotes the link address that along which a message should be transmitted if the destination is as specified in the corresponding `destination` field of the routing table entry as specified above. The links are addressed from number 0 to 1, the link addressing scheme may be different for different nodes.

The algorithm is a modified version of distance vector routing protocol. The difference being that in distance vector routing or RIP protocol the nodes continuously transmit the distance vectors to all the destinations. However in case of our problem statement we have an extra prior knowledge of the topology of the network (i.e mesh topology) so the only knowledge that is required is the positions of the faulty nodes, once we have their positions then any shortest path finding algorithm can be implemented on the resulting topology to route the message in the correct direction. Note that the computation of the shortest path is done at each node to improvise the path lengths within the duration of stabilization period of a fault. Assume that each node has a unique id defined as (for node $N[i][j]$):-

$$ID_{ij} = N \times i + j \tag{1}$$

Similarly if one knows the ID of the node than the corresponding position (i, j) can be computed as

$$i = \left\lfloor \frac{ID_{ij}}{N} \right\rfloor \; j = (ID_{ij})\%N \tag{2}$$

For the purpose of fault identification we introduce the following messages:-

`fault<`id_1**`,`** id_2**`, ...>`** : This message propogates the positions of the faulty nodes in the network. The nodes id_1, id_2, ... are all faulty nodes with their corresponding IDs. As described in equation **??** the position of the faulty nodes can be identified.

`recovered<`id_1**`,`** id_2**`, ...>`** : This messaeg signifies that the nodes with ids id_1, id_2, ... have recovered from a fault. Thus these nodes can now propogate the messages and should now therefore be included in the shortest path computations.

These messages are used in the algorithm for distributed control of routing.Here are steps taken by the algorithm:-

send of `fault<`id`>` message by node n :-

```
      list_id = {}
      for all node in neighbour(n) :
        if node is faulty:
          list_id = list + node
        end
      end
      if not list_id.isEmpty() :
        send fault<list_id> to all the non faulty negibours
        list_faulty.union(list_id)
      end
      update routing table
```

recieve of `fault<`$idlist$`>` message by node n :-

```
      list_id = idlist
      for all node in neighbour(n) :
        if node is faulty:
          list_id = list + node
        end
      end
      send fault<list_id> to all the non faulty negibours
      list_faulty.union(list_id)
      update routing table
```

send of `recovered<`id`>` message by node n :-

```
      recovered_id = {}
      for all node in neighbour(n) :
        if node is in list_faulty and node is not faulty:
          recovered_id  += node
        end
      end
      if not recovered_id.isEmpty() :
        send reovered<recovered_id> to all the non faulty negibours
        list_faulty.substract(recovered_id)
      end
      update routing table
```

recieve of `recovered<`$idlist$`>` message by node n :-

```
      recovered_id = idlist
      for all node in neighbour(n) :
        if node is in list_faulty and node is not faulty:
          recovered_id  += node
        end
      end
      send reovered<recovered_id> to all the non faulty negibours
      list_faulty.substract(recovered_id)
      update routing table
```

send of `M<`id`>` message by node n :-

```
      if id in routing table r[]
        send M<id> to r[id]
      if id in list_faulty:
        return ERROR
      g = grid_graph(N, N)
```

```
6       for all n in list_faulty:
7         g.remove_node(n)
8       end
9       sp = find_shortest_path(g, source=n, target=id)
10      nexthop = sp[1]
11      link_addr = link[sp[1]]
12      send M<id> along link_addr
```

recieve of `M<`id`>` message by node n :-

```
1       if id in routing table r[]
2         send M<id> to r[id]
3       if n == id:
4         return SUCCESS
5       if id in list_faulty:
6         return ERROR
7       g = grid_graph(N, N)
8       for all n in list_faulty:
9         g.remove_node(n)
10      end
11      sp = find_shortest_path(g, source=n, target=id)
12      nexthop = sp[1]
13      link_addr = link[sp[1]]
14      send M<id> along link_addr
```

The above routing algorithm uses a routine `find_shortest_path()`. This routine can take $O(|V|log(|V|))$ computational time for general topology however for square mesh topology there can be a better algorithm that computes the shortest path in $O(|V|)$ computation time.

`find_sortest_path(g, source, target)`:

```
1  s_x = (source)mod N ; t_x = (target)mod N
2  s_y = (source)/N; t_y = (target)/N
3  d = |s_x - t_x| + |t_y - s_y|
4  for n in list_faulty:
5    compute d' = |s_x - n_x| + |n_y - s_y|
6    if |mind(d) - d| > |d - d'|
7      mind = n
8  end
9  next hop is the neighbour along diagonal band from source to mind node
```

5 Analysis

5.1 Time Bounds

Since the algorithm borrows its basic ideas from the distance vector routing algorithm, it requires some stabilization time until the network state is changed to reflect the shortest path routes. In the distance vector routing the distance vectors are broadcasted in a periodic fashion thus requiring $O(d)$ where d is the diameter of the network. Now in the worst case $d = O(|V|)$, thus distance vector routing requires an stabilization time of $O(|V|)$.

The algorihtm proposed above guarentees that the message will be routed along the shortest path within the granuality if the message propogation delay for the `fault<`id`>` message. Since there are $N \times N$ nodes arranges in square grid, the worst case propogation of the message can require $\sqrt{2}N$ hops to propogate the entire network (i.e the case when on of the corner nodes gets faulty). Therefore in terms of number of nodes $|V|$ the information propogation delay is bounded by $\sqrt{2|V|}$ i.e $O(\sqrt{|V|})$. Let the maximum delay that can occur in the message transmittion across a link is T_{max}, thus the time required

by the algorithm to stabilize (T_{stab}) the network state in all the nodes can be given by:-

$$T_{stab} \leq \sqrt{2|V|} \times T_{max} = O(\sqrt{|V|}) \tag{3}$$

This value of T_{stab} gives a window of time in which the routing path can deviate from the shortest path. The time to live value in the comunication messages should therefore be bounded by $c_1 \times T_{stab}$ (where c_1 is a constant), because when a message is transmitted there should be enough time for the network to stabilize and then there should be enpug time for message to be transmitted along the shortest path. Now in the worst case the stabilization time is T_{stab} and the shortest path can be of length $|V|$ thus the time to live *ttl* for messages in the mesh topology is bounded by:-

$$ttl \geq T_{stab} + T_{max} \times |V| = T_{max} \times (\sqrt{2|V|} + |V|) = O(|V|) \tag{4}$$

This equation gives a bound on the max time required by the algorithm i.e. Time Complexity of the algorithm as $O(\sqrt{|V|})$ rounds for stabilization (fault tolerance) and $O(|V|)$ rounds for message propogation.

5.2 Message Complexity

In the naive distance vector routing algorithm there is a message transfered along each edge in both directions. Thus the message complexity of the distance vector routing protocol is given by $2 * |E| = O(|E|)$ messages in each time period. In the algorithm proposed however the messages are broadcasted only when there is a fault in the network. In the worst case when a fault occurs `fault<`id`>` message is transmitted twice along each link (once in each direction). Thus the message complexity is $2 * |E| = O(|E|)$ but since it is a square mesh each node has a degree of 4 (except the nodes in border who have a degree of 3). ths we can write:-
$|E| = \frac{\sum deg(v)}{2} \leq \frac{4*|V|}{2} = 2 * |V| = O(|V|)$

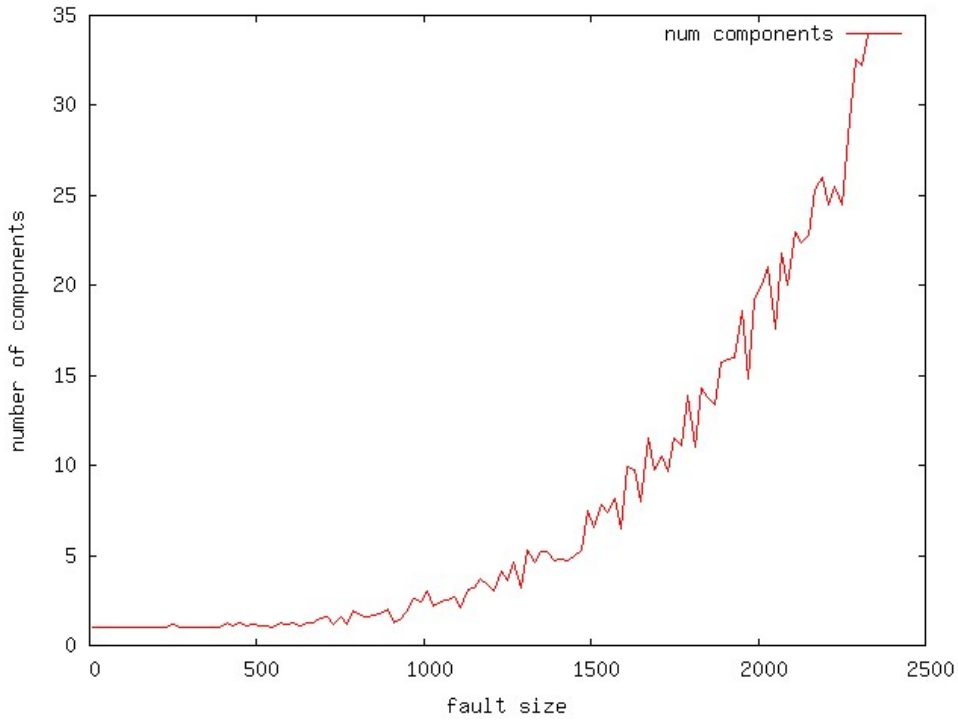
Thus the message complexity of the adaptive routing algorithm is $O(|V|)$ for the stabilization phase and $O(|V|)$ for the propogation phase. Although the message complexity of the algorithm is same as that of naive distance vector routing algorithm the actual number of messages required for stabilization are quite less. For example in a time interval of T secs the number of protocol bound messages for DVR protocol are $\frac{2|E| \times T}{T_{period}}$. For the modified algorithm let us assume that there is a constanct failure rate of λ per sec. Then:-
$E_T(failures) = \lambda \times T$
Number of messages $= 2|E| \times \lambda \times T$ which is very less as compared to DVR protocol given that the mean time to failure (MTTF) is less than T_{period}.

5.3 Message delivery

The above algorithm gets a complete snapshot of the network state and is thus tolerant to any number of faults in the mesh. The message will get delivered provided there is a route from the source to the destination i.e the source and the destiantion are in the same component. In the worst case the destination and source are always in two different components. For the graph to be divided in two components the number of failures $|F|$ is bounded as:-

$$|F| \geq N = \sqrt{|V|} \tag{5}$$

Thus the protocol can handle atleast $O(\sqrt{|V|})$ fault in the mesh topology. As evident from figure 5.3The average number of nodes that should be removed is considerably more than this bound.



6 Discussions

In the present work we have proposed an adaptive algorithm with completely distributed control to route the messages in the shortest paths (i.e. deviating only within a finite time interval). The algorithm proposed doesn't require back tracking like naive depth-first search and breadth first search algorithms. Althoug the algorithm proposed has been analysed on grid networks, it can be easily generalized for other topologies of on-chip interconnection networks. The algorithms provides admissible bounds on the time and message complexity, and can tolerate large number of faults. The routing time of the algorithm is also expected to be less as compared to other protocols in literature since always the shortest path is taken and there is no back tracking. The algorithm also doesn't depend on the properties of the fault i.e. convex or concave faults. This is thus a generalized fully distriburted adaptive algorithm for ensuring shortest path deliveries.

References

[1] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen. Dynoc: A dynamic infrastructure for communication in dynamically reconfugurable devices. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 153 – 158, 2005.

[2] Paul Bogdan, Tudor Dumitras, and Radu Marculescu. Research article stochastic communication: A new paradigm for fault-tolerant networks-on-chip, 2007.

[3] Pirretti Link Brooks, M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, and M. J. Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *In: IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design*, pages 46–51, 2004.

[4] Chun-Lung Chen and Ge-Ming Chiu. A fault-tolerant routing scheme for meshes with nonconvex faults. *Parallel and Distributed Systems, IEEE Transactions on*, 12(5):467 –475, 2001.

[5] M.-S. Chen and K.G. Shin. Adaptive fault-tolerant routing in hypercube multicomputers. *Computers, IEEE Transactions on*, 39(12):1406 –1416, 1990.

[6] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *Parallel and Distributed Systems, IEEE Transactions on*, 11(7):729 –738, 2000.

[7] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed. Hyperswitch network for the hypercube computer. In *Proceedings of the 15th Annual International Symposium on Computer architecture*, ISCA '88, pages 90–99, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[8] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[9] C.J. Glass and L.M. Ni. The turn model for adaptive routing. In *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, pages 278 –287, 1992.

[10] Amir Hosseini, Tamer Ragheb, and Yehia Massoud. A fault-aware dynamic routing algorithm for on-chip networks. In *ISCAS*, pages 2653–2656, 2008.

[11] C-K. Kim and D. A. Reed. Adaptive packet routing in a hypercube. In *Proceedings of the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues - Volume 1*, C3P, pages 625–630, New York, NY, USA, 1988. ACM.

[12] T. Lehtonen, P. Liljeberg, and J. Plosila. Fault tolerant distributed routing algorithms for mesh networks-on-chip. In *Signals, Circuits and Systems, 2009. ISSCS 2009. International Symposium on*, pages 1 –4, 2009.

[13] Ming Li, Qing-An Zeng, and Wen-Ben Jone. Dyxy - a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 849 –852, 2006.

[14] R. Libeskind-Hadas and E. Brandt. Origin-based fault-tolerant routing in the mesh. In *High-Performance Computer Architecture, 1995. Proceedings., First IEEE Symposium on*, pages 102 –111, 1995.

[15] Muhammad Ali Michael, Michael Welzl, Martin Zwicknagl, and Sybille Hellebr. Considerations for fault-tolerant network on chips, 2005.

[16] Alan Olson and Kang G. Shin. Fault-tolerant routing in mesh architectures. *IEEE Transactions on Parallel and Distributed Systems*, 5:1225–1232, 1994.

[17] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C.R. Das. Exploring fault-tolerant network-on-chip architectures. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 93 –104, 2006.

[18] Zhen Zhang, A. Greiner, and S. Taktak. A reconfigurable routing algorithm for a fault-tolerant 2d-mesh network-on-chip. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 441 –446, 2008.