Dynamic Fault Tolerance with Misrouting in Fat Trees

Frank Olaf Sem-Jacobsen
Department of Informatics
University of Oslo
Postboks 1080 Blindern
0316 Oslo, Norway
frankose@ifi.uio.no

Tor Skeie, Olav Lysne
Networks and Distributed Systems
Simula Research Laboratory
P.O.Box 134
1325 Lysaker, Norway
{tskeie,olavly}@simula.no

José Duato
Dept. de Ingenieria de Sistemas, Computadores y Automatica
Universidad Politecnica de Valencia
Pob 22012,46071 - Valencia
Spain
jduato@gap.upv.es

Abstract

Fault tolerance is critical for efficient utilisation of large computer systems. Dynamic fault tolerance allows the network to remain available through the occurance of faults as opposed to static fault tolerance which requires the network to be halted to reconfigure it. Although dynamic fault tolerance may lead to less efficient solutions than static fault tolerance, it allows for a much higher availability of the system. In this paper we devise a dynamic fault tolerant adaptive routing algorithm for the fat tree, a much used interconnect topology, which relies on misrouting around link faults. We show that we are guaranteed to tolerate any combination of less than $\frac{\text{num\_switch\_ports}}{2}$ *link faults without the need for additional network resources for deadlock freedom. There is also a high probability of tolerating an even larger number of link faults. Simulation results show that network performance degrades very little when faults are dynamically tolerated.*

1. Introduction

As the size and complexity of computer systems increases, efficient fault tolerance methods become of increasing importance. The key to high performance in today's supercomputers and high-end servers is parallelism, interconnecting a large number of processing units which work together. The interconnection network interconnecting the processing units among themselves and/or these units with the storage subsystem becomes a critical component in the system, and it is essential that it is able to operate correctly despite intermittent failure of network elements.

To keep the network connected despite the failure of switches and links, several strategies have been proposed and/or implemented. The most frequently used technique in commercial systems is to implement the circuits required to switch off the faulty component (e.g. switching off several planes of the BlueGene/L 3D torus), possibly switching in some spare components (e.g. by implementing a few extra planes). This strategy has the benefit of simplifying routing by keeping the same topology, but it is quite expensive and often switches off too many healthy components. Another approach followed in many academic papers consists of exploiting the existence of alternative paths in the network by defining a fault-tolerant routing algorithm. These algorithms provide alternative paths that can be followed in case of failure. They have become very popular among researchers because they do not require the use of spare components and usually disable very few (if any) healthy components. We will refer to this strategy as dynamic fault tolerance in this paper. Endpoint dynamic fault tolerance allows the source nodes to choose a different path through the network up on being informed of a network fault, the approach used in e.g. [15]. Local dynamic fault tolerance, on the other hand, provides multiple paths from any single network element to avoid any faulty component it may be connected to. Unfortunately, most proposals introduce significant complexity (e.g. several additional virtual channels) to avoid deadlock in the presence of failures.

The third and most recent group of techniques consists of adding a network management layer that explores the topology in case of failure and computes new routing tables. This is by far the most flexible approach, supporting a large number of failures. This approach, known as network reconfiguration, can be considered as an extension of the network configuration software found in technologies like Myrinet and InfiniBand. However, flexibility comes at the expense of using a generic routing algorithm that may not be optimal for a given topology. Also, statically reconfigur-

ing the network is time-consuming since the network applications must be halted and the network drained of all traffic to avoid dependencies between packets routed following the old and new routing algorithms. Dynamic fault tolerance is much more time efficient in this respect, both the endpoint, but even more the local, approaches. However, the local solution of routing around the faulty elements may lead to nonoptimal paths. Despite this, local dynamic fault tolerance is overall much more efficient in large systems with frequent fault events, and it is the approach we will follow in this paper.

Rerouting packets around network faults is most easily achieved by using an adaptive routing algorithm. An adaptive routing algorithm supplies several alternative paths for a destination if available, allowing the switch to choose the output queue with, for instance, the shortest length. Related to fault tolerance, when discovering a faulty network element, the switch is free to adaptively choose a different path which is not affected by the fault. It is therefore relatively straightforward to provide dynamic fault tolerance in networks with multiple available paths such as meshes and tori topologies.

Interconnection networks are often interconnected in regular topologies like the k-ary n-cube, mesh, and various Multistage Interconnection Networks (MIN). Multistage interconnection networks were first introduced in 1953 by C. Clos as a means of producing nonblocking switching networks consisting of few switches organised in switching stages [4]. The fat tree, a MIN developed by C. Leiserson in 1985 [12], is an ideal topology for use in supercomputer systems. It is an area universal network, a network able to emulate any other network topology built from the same amount of hardware with only a small decrease in computational efficiency [12]. Because of this the fat tree is a much used interconnection topology in parallel computer systems. As many as seven of the top 20 supercomputers rely on the fat tree topology in some parts of their interconnection network (Q4, 2005). A classical example is CM5 [5], and a more recent example is SGI Altix 3700 [21] which is used in NASA's Colombia, currently ranked number three. Most of the commercial interconnects for SANs and clusters also use fat trees or other similar bidirectional MINs, for instance InfiniBand [1], Myrinet 2000 [2], and Quadrics [17].

The fat tree (Figure 1) is usually a tree with multiple roots. The bottom stage of the network consists of the processing units connected as leaves to the switches at the bottom of the tree. The switches are organised in stages, and the number of stages depends on the number of nodes connected to the network and the switch

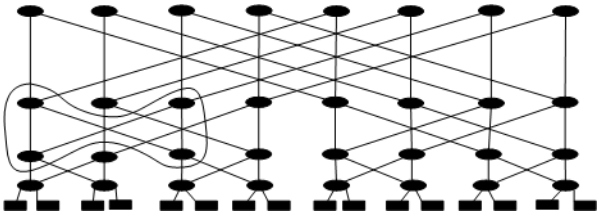


Figure 1. A small fat tree with radix 4 switches. The black dots are switches, and the black squares are processing nodes connected to the bottom of the network. The outline describes a switch group as it is defined in Section 3.

radix (the number of ports in the switch). The fat tree is distinct from an ordinary tree in that the aggregate link capacity between two switch stages is maintained at every stage to the top, i.e. the branches of the tree get fatter nearer the root. Fat-trees in which the capacity of the links themselves increases towards the root are also feasible, but the common approach is to increase the number of links and switches at the upper stages. The fat tree we consider in this paper is the commonly used K-ary N-tree as it is defined in [16].

Packet routing in the fat tree is performed as in any tree topology.The packet is forwarded from the source upwards to any least common ancestor of the source and destination, and then downwards following a deterministic path determined by the least common ancestor to the destination. We call this the upward phase and the downward phase.

Previous approaches to dynamic fault tolerance in fat trees [19, 18] have considered dynamic fault tolerance combined with deterministic routing. Strict routing rules, extra mechanisms in the switches, and additional network resources have been required to route around faults and guarantee deadlock freedom. In this paper we will achieve local dynamic fault tolerance in conjunction with adaptive routing. However, since the fat tree is deterministic in the downward phase, the use of an adaptive routing algorithm will not automatically lead to local dynamic fault tolerance. We therefore develop an adaptive local dynamic fault tolerant routing algorithm which misroutes packets one hop towards a different destination to reach an alternative deterministic path, avoiding the fault.

The main contributions of this paper are: 1) A new and efficient routing algorithm for fat trees that is able to tolerate at least up to an including $\frac{radix}{2} - 1$ link failures without disconnecting the network; 2) a detailed theoretical proof of the fault tolerant properties of the proposed routing algorithm; and 3) a performance eval-

uation showing that network performance experiences a small degradation in the face of faults.

After an overview of past efforts in the field of fault tolerance in interconnection networks in Section 2, we propose an adaptive routing algorithm designed to provide dynamic fault tolerance in fat tree networks in Section 3. In Section 4 we present an evaluation of the proposed algorithm, and a conclusion is given in Section 5.

2. Related work

Not all MIN topologies intrinsically provide multiple paths that may be utilised for fault tolerance. For example, the Butterfly network [9] only provides a single path between each source/destination pair. A large amount of work concerning fault-tolerant MINs is based on adding additional hardware in terms of adding extra links and switches to existing switching stages, or adding entirely new switching stages [23]. Another much used approach is to route the packet through the network in several passes. Routing through multiple passes requires the MIN to possess a dynamic full access (DFA) property, i.e. the ability to route between any node pair using a limited number of passes [10]. Hybrid approaches combining multiple paths and multiple passes have also been suggested [22]. In [13] the authors give an overview of a number of fault-tolerant MIN topologies, describe their basis for fault-tolerance and evaluate their performance. J. Sengupta and P.K.Bansal [20] use several parallel MINs to create redundancy, with each MIN functioning as a disjoint path between the sources and destinations. The fault tolerance of Butterfly networks is studied in [3] and [11]. F.Cao and D. Du [3] discuss the fault tolerance property of the Butterfly network and show how to construct paths through the faulty network. In [7] the authors analyse the fault tolerance properties of MINs without additional redundancy. Most of his work is based on unidirectional MINs, there is little work done on bidirectional MINs.

None of the above contributions consider dynamic fault tolerance. An exception to this statement is the approaches in [19] and [18]. The first paper describes a method for achieving dynamic fault tolerance using two parallel fat-trees with crossover paths between the corresponding switches in both trees. The method is shown to yield almost as good performance as when using static fault tolerance to route around failed links. The second paper presents a dynamic fault tolerant routing algorithm which provides dynamic one fault tolerance in singleton fat-trees. A deterministic routing algorithm is used combined with a dynamic fault tolerance algorithm, so the use of additional virtual layers is therefore required to guarantee deadlock freedom.

Concerning fat-trees, some work has been done on fat-trees in general, and to some small extent on fault tolerance. Fat-trees are extended to generalised fat trees in [14], allowing them to have a varying number of switches in switching stages and links between the stages. Another variation of fat-trees are orthogonal fat-trees as presented in [24]. Orthogonal fat-trees are built using simple two level fat trees as the building blocks to achieve a larger network and they provide only a single path between any source/destination pair. The authors extend this work in [23] by providing several disjoint paths through the addition of network links.

3. Fault-Tolerant Routing in the Fat Tree

We consider a fat tree made up of virtual cut-through switches. In the virtual cut-through switching scheme, packets are divided into small data units. During traversal of the network, the separate units of a packet may be spread over several switches, but on being blocked the packet as a whole is buffered in a single switch, thus reducing contention and simplifying deadlock avoidance.

Failures may be transient or permanent. Transient failures can be easily handled by using a suitable communication protocol that re-transmits the information in case of transmission errors. Therefore, we will focus on permanent failures.

We consider only link faults in this paper. A link is either present and operational, or faulty, in which case it may be viewed as non-existent. We assume that the network elements directly connected to the failed element can detect the failure of that element, information about failed elements need not be propagated further. Additionally, we do not allow the links connecting processing nodes to the bottom of the tree to fail. Note that additional hardware is required to support the failure of these links (e.g. using two NICs per processing node to connected to two different network ports, which also implies doubling the number of network ports).

In the first section below we will present a routing algorithm which is able to guarantee a connected network for one link fault without modifying the packet header or network switches in any way, and for less than $\frac{radix}{2}$ link faults if we add a small field of length $\frac{radix}{2}$ bits. The non-minimal paths used to avoid the faulty links may lead to deadlocks, a state where every next hop

queue for a set of packets is full, forming a cyclic dependency. We will therefore present a proof that shows that the proposed algorithm is deadlock free in Section 3.2.

3.1. The Routing Algorithm

The routing algorithm we present here relies on the numerous paths between two neighbouring switches. When a link between two switches fails, the switch at the top of the failed link will no longer have a path to the destinations it used to reach through the failed link. However, the switch at the bottom end of the failed link has links to numerous switches next to the switch at the top of the failed link. If a packet is routed to one of these switches it will have a valid path down to its destination. We construct a routing algorithm that misroutes the packet, forcing it to traverse two extra hops, one hop down and one hop up, to reach a fault-free path to the destination upon encountering a link fault. The path around a faulty link is shown in Figure 2, the details of this will be discussed later.

Definition 1. *An adaptive routing function R* supplies a set of output queues to be used by a packet in switch s to reach a destination d.*

We construct an adaptive, dynamic fault-tolerant routing algorithm by dividing packet routing into the following two phases. In the following, a faulty queue is a queue associated with a faulty link.

1. In the upward phase, all upward queues are supplied by the routing function and one is selected for output. If any of the queues are faulty, they are simply not chosen.
2. In the downward phase only one single queue is supplied by the routing algorithm, the shortest path from the current switch to the destination. If this queue is faulty, all other downward queues are supplied, forcing the packet to be misrouted through one of them. Once a packet has been misrouted downwards, a new upward phase commences, with the adaptive algorithm excluding queues associated with the link on which the packet arrived.

Definition 2. *A* U-turn switch *is a switch in which a transition from a downward to upward phase takes place.*

There will be U-turn switches in the network only in the case of faults.

Definition 3. *A* U-turn *is a transition from downward to upward movement involving a downward link, a U-turn switch, and an upward link.*

Thus the term U-turn does not apply to the upward to downward transition which separates the upward and downward paths in the routing algorithm. In a fault-free network no U-turns will therefore ever be performed.

This simple algorithm will be able to guarantee network connectivity when no more than one link fails [18], without having to modify the packet header in any way. For it to be able to tolerate a larger number of link faults, the algorithm must systematically test the set of available paths once it has finished misrouting downwards and is about to perform the U-turn and commence on a new upward phase. The reason is that several of the other switches reachable upwards from the U-turn switch might not have a fault-free downward link towards the destination. It might therefore be necessary to test several upward paths from the U-turn switch to find one that leads one switch stage closer to the destination. To preserve the adaptivity required by the routing algorithm in the upward phase we must implement a systematic check such that all possible paths are tested without forcing the packet on to any path unless we know that it is deadlock free. We will show below that this may be achieved by having a field in the packet header, a misroute vector, of length $\frac{radix}{2}$ bits, one for each of the upward paths to test. Combined with the first algorithm the adaptive dynamic fault-tolerant algorithm which is guaranteed to tolerate up to and including $\frac{radix}{2} - 1$ link faults is as follows:

1. In the upward phase, all upward queues are supplied by the routing function and one is selected as the packet's output. If any of the queues are faulty, they are simply not chosen.
2. In the downward phase only one single queues is supplied by the routing algorithm, the shortest path from the current switch to the destination. If this queue is faulty, all other downward queues are supplied, forcing the packet to be misrouted through one of them.
3. A switch receiving a packet from a link connected to an upper stage for which it has no downward path sets the bit in the misrouting bit-vector in the packet header corresponding to the link on which the packet arrived (it is not necessary to try to use this link for forwarding the packet since we know the path is broken). This switch is a U-turn switch.
4. The U-turn switch adaptively chooses one of its upward queues which does not have its corresponding bit set in the packet header and subsequently inserts the packet into this queue.

5. If the new switch that the packet arrived at from the U-turn switch has a valid downward path as the next hop, the misrouting vector in the packet header is reset and the packet is forwarded as normal. Otherwise the packet is misrouted back to the U-turn switch down the same link on which it arrived at the current switch and step three is repeated. If this link has just failed reset the misroute vector and repeat from step 2.
6. A U-turn switch receiving a packet from an upward link in which all other bits in the misrouting vector are set discards the packet, R_m is disconnected. The switch may inform the source that the path is no longer available. We know we have tested all upward links of the U-turn switch and there is no available path to the destination from this switch.

This routing algorithm will be known as R_m throughout the paper. We will now present a series of arguments and proofs to show the validity, and explore the properties and limits of R_m.

Definition 4. *Two switches are* neighbours *if they are interconnected through a single link.*

Definition 5. *Two sets of switches, a and b, are* completely interconnected *if every switch in a is a neighbour of all switches in b (and implicitly vice versa).*

Definition 6. *A* switch group g *is the union of two completely interconnected sets of switches, a and b, where a contains only switches at stage l and b contains only switches at stage $l+1$. a contains all neighbours at stage l of all switches in b, and b contains all neighbours at stage $l+1$ of all switches in a, $g = a \cup b$.*

Given the interconnection pattern of a fat tree, each switch group consists of $\frac{radix}{2}$ upper stage switches and $\frac{radix}{2}$ lower stage switches. Every switch in the fat tree is part of two switch groups, but the upper stage of one switch group, and at the lower stage of another. The switches at the topmost and bottommost stages of the tree are only members of one switch group. Figure 1 shows a fat tree consisting of switches of radix four. The switch group encompassed by the line therefore consists of two upper stage and two lower stage switches. For simplicity it is assumed that the top stage switches of the fat tree have the same radix as the other switches in the network, but with $\frac{radix}{2}$ unused ports. Practical implementations may use switches without the unused ports, or use the extra ports to halve the number of top stage switches.

Corollary 1. *There are $\frac{radix}{2}$ disjoint shortest paths between any two switches at the same stage in a switch group.*

Proof. It follows from Definition 6 that any switch s at the upper/lower stage of a switch group has $\frac{radix}{2}$ neighbours at the lower/upper stage. Each of these neighbours are neighbours to all switches in the group at the same stage as s. Thus, there exists $\frac{radix}{2}$ paths from s to each of the other switches in the group of the same stage, one through each of its neighbours in the group. Since there are no links between switches at the same stage these paths will be the shortest paths, each of length two hops. □

Lemma 1. *R_m is connected when there are less than $\frac{radix}{2}$ link faults.*

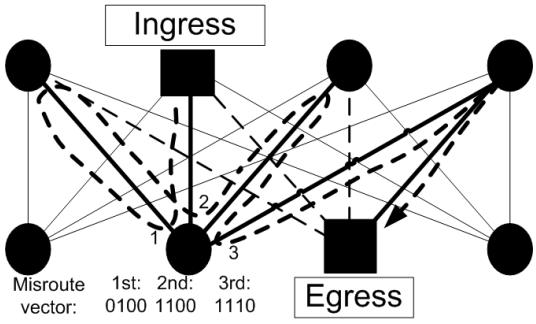
Proof. In the upward phase R_m is obviously always able to forward every packet one stage upwards with less than $\frac{radix}{2}$ link faults. As long as one link is available in the upward direction the packet may be forwarded. The downward phase is as follows: With $\frac{radix}{2} - 1$ link faults in the system at least one of the $\frac{radix}{2}$ upper stage switches in a group with link faults will not be connected to any faulty link. We call this switch S_t. From corollary 1, there are $\frac{radix}{2}$ disjoint paths between any two switches at the same stage in a group. $\frac{radix}{2} - 1$ link faults is not enough to disconnect all these paths, and thus, any upper stage switch connected to a faulty link is able to reach S_t, which we know has a healthy link moving the packet one hop closer to its destination. Each lower stage switch in the group will be connected to S_t, so by misrouting to an arbitrary lower stage switch S_b there will be a path to S_t. By checking all upward links from S_b exactly once, we are guaranteed to reach S_t without cycles. Once the packet has reached a lower stage switch in the group with a valid downward path it enters a new group, so subsequent link failures encountered will be tolerated in a different group, further down in the tree with at most $\frac{radix}{2} - 2$ link faults. □

In addition to Lemma 1 which proves that the algorithm dynamically tolerates faults, we must show that the algorithm is livelock free. The key to this is the misroute vector.

Lemma 2. *R_m is livelock free when there are less than $\frac{radix}{2}$ link faults in the network.*

Proof. As the number of links is finite, a livelock requires that a packet is forwarded in a loop. There must therefore exist a set of switches that the packet traverses an unlimited number of times. There are obviously no such loops in the upward phase, the only possible cause of a loop is the U-turn performed when misrouting around a fault. However, the misrouting vector guarantees in point 4 of the misrouting algorithm that

all the upper stage switches in the switch group containing the fault are visited at most once for each link fault in the group, and hence none of these may be involved in a livelock. Since there are less than $\frac{radix}{2}$ link faults we are guaranteed a path which moves the packet out of the switch group and one stage lower down towards the destination. Consequently, every time a packet is misrouted, it will find a path which brings it one stage closer to its destination, and the algorithm is livelock free. □



(a)

Figure 2. Combinations of $\frac{radix}{2} - 1$ faults with misrouting within a group in radix=8 network. The terms ingress and egress refer to the switches where the packet enters and leaves the switch group respectively. The bold lines are the links traversed by the packet, the dashed links are a faulty, and the long dashed path with an arrow is the path followed by a packet. The numbered U-turns refer to the corresponding states of the misroute vector (listed next to the U-turn switch) as the U-turn is performed.

Figure 2 gives an example of a fault configuration with $\frac{radix}{2} - 1$ faults and how it is tolerated by the routing algorithm. The figure depicts a switch group in a network consisting of radix eight switches. The packet has to attempt to use several upward paths from the lower stage switch in order to reach an upper stage switch with a fault-free path to the destination. It is clear from this figure that there exist combinations of a large number of link faults that allow the routing algorithm to remain connected. Therefore, in addition to be able to guarantee $\frac{radix}{2} - 1$ fault tolerance, the algorithm will with some probability be able to tolerate an even larger number of arbitrary link faults.

3.2. Deadlock Freedom

We have shown that R_m is connected and livelock free for less than $\frac{radix}{2}$ link faults. Next we will prove that it is deadlock free as long as we can guarantee connectivity. The non-fault tolerant upward phase, downward phase routing algorithm relies on that there cannot be any downward to upward turns in the network to be deadlock free. However, when we misroute around a fault we introduce downward to upward turns and create dependency cycles that may cause deadlocks. Deadlock freedom has previously been provided by utilising virtual channels to break the cycles [18]. In this section we will show that such a solution is unnecessary when we use adaptive routing since the cycles will not cause deadlocks. For this analysis, we assume the use of virtual cut-through switches and will focus on the dependencies between the switch queues where the packets are buffered.

The following definition and theorem are taken from [6]:

Definition 7. *A routing subfunction R1 of the routing function R* is a routing function connecting the same set of sources and destinations as R*, but provides a subset of the queues provided by R*.*

R1 may in other words be viewed as a limited version of R* where some of the network queues supplied by R* for a destination are not supplied for that destination by R1.

Theorem 1. *A routing algorithm is deadlock free iff there exists a connected routing subfunction R1 of the routing function R* which has no cycles in its extended channel dependency graph.*

The theorem relies on the terms dependency, cross dependency, and extended channel dependency graph as they are defined in [6]. There is a dependency from one queue to another if the first packet in the first queue is headed to the second queue following the routing algorithm. There exists a cross dependency when there are packets in queues that are legal for R*, but not for R1, which have a queue that is legal for R1 as their next hop. The extended channel dependency graph is a graph whose vertices are queues, and the arcs are between pairs of queues where the first queue is either dependent or cross-dependent on the other. The extended channel dependency graph is similar to the ordinary channel dependency graph except that the latter only considers queues that are dependent, not cross-dependent.

Theorem 1 is primarily intended for direct networks, so in order to make use of Theorem 1 we must consider our indirect fat tree as a direct network. This im-

plies that every node in the network, both processors and switches, must be viewed as packet producers, consumers, and packet forwarders. This creates a hugely complex routing algorithm since many of the paths in the network will contain U-turns when packets are sent from one switch to another. We may however disregard many of these U-turns when considering deadlock freedom.

A configuration is an assignment of a set of packets to the queues in the network, and a legal configuration is such an assignment that is reachable from an empty network by injecting and forwarding packets as the routing algorithm dictates. A deadlocked configuration is a nonempty legal configuration where several of the network queues are full, and all queues supplied by the routing function for the first packet in every full queue are also full. Consequently, an illegal configuration which is deadlocked is not an issue since it may never be reached. It is therefore only necessary to consider legal configurations when analysing the dependencies of the routing algorithm. In an indirect network such as the fat tree a switch will never be a destination, and thus all configurations involving U-turns without the U-turn switch being due to a faulty link are illegal. This eliminates the numerous U-turns that would occur if packets should be transmitted between two switches in the network. We are left with U-turns only occurring below faults. In fact, the only thing we must consider which is different from a direct network is that the routing subfunction must be connected with respect to any packet in any queue in the network, i.e. we must view every switch as a "packet producing" node, but not as a destination.

The deadlock freedom of R_m is not obvious since many of the downward to upward turns performed when attempting to misroute around a fault may close cycles in the channel dependency graph. To show that R_m is deadlock free we must show the existence of a routing subfunction R'_m of R_m which is connected and has no cycles in its extended channel dependency graph as is required by Theorem 1. It is important to note that the routing subfunction need not be implemented in any way, it is sufficient that we can prove the existence of such a function. The rest of this section is devoted to finding R'_m and proving that it is cycle free in its extended channel dependency graph and connected within the fault tolerance limits set by the dynamic fault-tolerant routing function R_m. In order to achieve this we must create a basis upon which to construct the routing algorithm. We will first identify which U-turn switches may be part of a deadlock.

Definition 8. *Assume we have a channel dependency cycle. A switch is* part of a dependency cycle *when queues in that switch are part of the channel dependency chain forming the cycle in the extended channel dependency graph.*

Lemma 3. *A dependency cycle must always involve two or more U-turn switches.*

Proof. Suppose there is a dependency cycle involving only one U-turn switch, b, with the downward link of the U-turn from switch a to b and the upward link of the U-turn from b to switch c. In this case there is a chain of dependencies from b, c to a, b which is closed by the downward to upward U-turn a, b, c. In other words, it is possible to follow the chain of dependencies from b, c and arrive at a, b. However, if we follow the chain of dependencies from the upward link it will lead us some way upwards in the tree before it will turn downwards. a and c can not be part of the same subtree (otherwise switches at higher stages would have more than one path downward to a destination, which is not the case) so they have no common ancestors, and it is impossible to reach a from c with only one upward and downward phase. From here on the chain will therefore only involve downward channels since there are no other U-turns and the chain will terminate in a processing node. Thus, there is no cycle. □

As we shall see, not all combinations of U-turn switches may form a dependency cycle, it depends on where in the fat tree the U-turn switches are located.

Definition 9. *The* subtree of a switch *consists of all the links and switches reachable in the downward direction from that switch.*

Every switch is member of a specific set of subtrees, i.e. there is a certain set of tree roots that can be reached in the upward direction from the switch. A top rooted subtree is a subtree of a top stage switch.

Definition 10. *Two switches are* members of the same set of top rooted subtrees *when they have all top stage subtree roots in common.*

Lemma 4. *Only U-turn switches that are members of the same set of top rooted subtrees may be part of a dependency cycle.*

Proof. First, a dependency cycle must involve two or more U-turn switches (Lemma 3). Next, consider the U-turn switches u and $v_1, v_2, \ldots, v_n$. Suppose that we have a deadlocked configuration and that u is member of a top rooted subtree st of which v_i is not a member for every $0 < i \leq n$. Any packet forwarded upwards through st from u will necessarily never reach v_i since v_i is not in st. Consequently, there is a path out of the cyclic dependency between the U-turn switches and thus no dependency cycle. □

From Lemma 4 we may make the following observation.

Observation 1. *A dependency cycle may only occur between U-turn switches at the same switch stage in the network since U-turn switches at different stages will be members of different sets of subtrees.*

The basis for observation 1 is that with two switches at different switch stages the switch at the upper stage will necessarily be a member of fewer subtrees than the switch at the lower stage, and they will therefore not be members of the same set of top rooted subtrees.

We have established that any deadlock in the network must involve at least two U-turn switches at the same switch stage. Therefore, if we can guarantee that a packet misrouted through a U-turn at stage l is forwarded in a subtree with no further failures in the links of the same stage l, it will not be able to close any dependency cycles.

Definition 11. *A* subtree which is fault-free at stage l *is a tree whose root is at the top stage of the fat-tree, and whose leaves are at stage l. The fault-free subtree contains excactly all nodes and links that are reachable through downward-links from the root, and that are at or above level l.*

Lemma 5. *A U-turn switch at stage l will not be part of any dependency cycle in the extended channel dependency graph if it is connected to a subtree which is fault-free at stage l and forwards misrouted packets in this subtree.*

Proof. For every link fault in the network both the faulty link and the downward link of the associated U-turn will be located in the same subtree. There are consequently no downward links of U-turns at stage l in the subtree that is fault free at stage l. Therefore, if we follow the possible chain of dependencies from the upward link of a U-turn through the subtree that is fault-free we will never encounter a downward link of a U-turn at stage l since we encounter no faults. Since we may only have cyclic dependencies between U-turns at the same stage and this is impossible, we are not able to create a cyclic dependency. □

These are the foundations necessary to construct our routing subfunction. The steps of the routing subfunction R'_m is listed below:

1. Adaptively forward the packet upwards towards any least common ancestor.
2. Upon reaching the least common ancestor forward the packet downwards.
3. If a fault is encountered in the downward direction misroute the packet one step downwards, to stage l.
4. Forward the packet back up the subtree that is fault-free at stage l.

R'_m is obviously connected when considering every switch as a packet producer. Any switch may forward packets upwards to a least common ancestor of any node pair, or forward the packet downwards if that is the shortest path to the destination, and any switch may forward packets downwards one hop when encountering a fault as long as there is less than $\frac{radix}{2}$ link faults. It is equally obvious that R'_m is a routing subfunction of R_m. The only limitation of R'_m is which upward path may be taken after a U-turn.

The deadlock freedom of R'_m will be proven below, but we must first determine the number of link faults for which we can guarantee that every possible U-turn switch is connected to a fault-free subtree.

Every possible U-turn switch in the network, that is, all switches except the top stage switches, has $\frac{radix}{2}$ upward links, each connecting them to a set of subtrees. The size of this set will depend on at which stage the individual switch is located. It is easy to see that if every one of the $\frac{radix}{2}$ upward links of a switch should fail there will be a link fault in all of the subtrees to which the switch is connected. Similarly, if there are only $\frac{radix}{2} - 1$ link faults in the network it is impossible to place a link fault in each of the subtree sets to which a switch is connected. This is the basis for Lemma 6.

Lemma 6. *Every possible U-turn switch is connected to a fault-free subtree of the same stage when there are less than* $\frac{radix}{2}$ *link faults in the network.*

Proof. Every U-turn switch in the network is connected to $\frac{radix}{2}$ subtree sets. To put one fault in every subtree set requires at least $\frac{radix}{2}$ link faults. One of the subtree sets will therefore consist of subtrees that are fault-free at the stage of the U-turn switch. □

It is apparent that the U-turns performed in a U-turn switch will not cause cyclic dependencies if the upward link of the U-turn is part of a subtree that is fault-free at the same stage. Furthermore, we know that with less than $\frac{radix}{2}$ link faults every U-turn switch will be connected to at least one subtree that is fault-free. We will now use this to prove that the routing subfunction R'_m is has no cycles in its extended channel dependency graph.

Theorem 2. R'_m *is connected and has no cycles in its extended channel dependency graph with less than* $\frac{radix}{2}$ *link faults in the network.*

Proof. With less than $\frac{radix}{2}$ link faults every U-turn switch is connected to at least one fault-free subtree (Lemma 6). A packet performing a U-turn in a U-turn switch at any switch stage will not be part of any dependency cycle involving U-turn switches at other stages (Lemma 4), or U-turn switches at the same stage when forwarded in the fault-free subtree from the U-turn switch (Lemma 5). Thus, none of the U-turns possibly performed by a packet on its path from its source to its destination will create dependency cycles in the extended channel dependency graph.

Any switch can reach any processor in either the upward or downward direction. In the upward phase R'_m follows the same algorithm as R_m and it is therefore connected. In the downward phase, with less than $\frac{radix}{2}$ link faults any packet encountering a fault at any stage will be able to reach a U-turn switch. From this it can reach a fault-free subtree which brings it at least one stage closer to the destination. The routing algorithm is therefore connected. □

Therefore, using Theorem 1, we conclude that R_m is deadlock free.

4. Performance Evaluation

The fault-tolerant routing algorithm has been evaluated through simulations in an event-driven simulator developed in-house at the Simula Research Laboratory, based on j-sim [8]. We have evaluated the effect of the number of link faults on the throughput and latency experienced by network traffic with a uniform distribution of packet destinations. The network is run both saturated and unsaturated to show the maximum achieved throughput and typical latency respectively. The switch radix is eight. The results are displayed in Figures 3(a) and 3(b). We have tested 500 random link fault combinations for every number of link faults between one and 20. We have also simulated the network with zero link faults for reference. We have only included results from the runs that neither deadlocked nor disconnected the network. The network has a width of 16 switches and a height of three switch levels not counting processing nodes, a 4-ary 3-tree. The vertical line in the plots marks the transition from the guaranteed fault tolerance region which goes from zero up to and including $\frac{radix}{2} - 1 = 3$ link faults, to the statistical region where there is only a probability of tolerating faults, from four link faults and upwards. In this region the probability of deadlock or disconnecting the network will gradually increase as the number of link faults increases.

As the number of link faults increases, the saturated throughput shows a steady decrease of about 2.7% per link fault to begin with, decreasing to about 2% decrease per link fault around 20 faults. Similarly, the saturated latency experienced by the packets increases to almost double that of zero link faults when there are 20 link faults in the network. This is in contrast to the unsaturated throughput which remains stable until we reach five link faults, from where it gradually starts decreasing with an increasing rate. As we approach 20 link faults the rate of reduction has increased to about the same as for the saturated network, indicating that the remaining paths that are not disabled by the faults have saturated. The unsaturated latency displays the same behaviour as the saturated latency, but it has a much lower starting point.

5. Conclusion

The fat tree is well-suited for interconnecting nodes in parallel supercomputers and clusters. It supports high throughput and has a large number of disjoint paths between any source/destination pair making it ideal for static fault tolerance. However, as static fault tolerance has quite a long turnaround time when tolerating a single link fault, we have proposed a dynamic fault tolerance algorithm utilising a simple misrouting mechanism to tolerate up to and including $\frac{radix}{2} - 1$ link faults locally. By relying on adaptive routing we are able to guarantee deadlock freedom for any link fault combination for which the fault tolerance algorithm can guarantee a connected network. We do not rely on the use of any network mechanisms such as virtual channels in order to achieve this. Simulations show that network throughput only suffers a small degradation when tolerating up to $\frac{radix}{2} - 1$ link faults dynamically, and beyond.

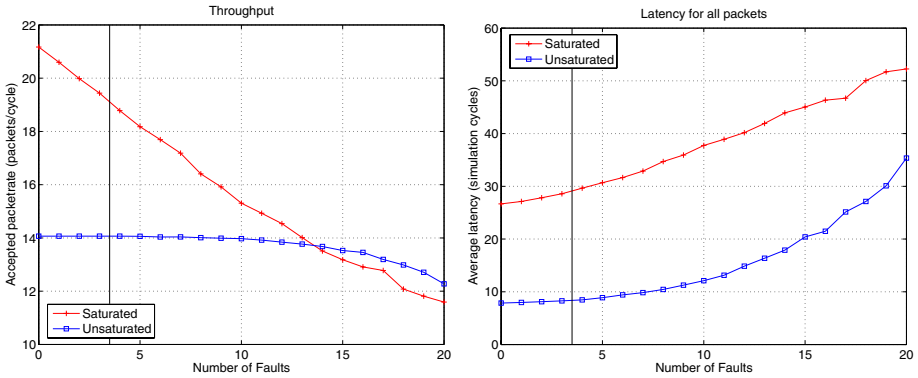
References

[1] I. T. Association. *InfiniBand Architecture Release 1.0, Volume One - General Specifications*, final edition, October 24th 2000.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[3] F. Cao and D.-Z. Du. Fault-tolerant routing and multicasting in butterfly networks. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 455–460. ACM Press, 1999.

[4] C. Clos. A study of non-blocking switching networks. *Bell Syst. Tech. Journal*, 32(2):406–424, March 1953.



(a) Throughput (b) Latency

Figure 3. Performance evaluation with link faults, both with a saturated (packet insertion rate is 32 packets/cycle) and unsaturated network (packet insertion rate is 14 packets/cycle). The vertical line marks the transition from guaranteed fault tolerance on the left side to probable fault tolerance on the right side.

[5] T. M. Corporation. Connection machine cm-5 technical summary. Technical report, 1993.

[6] J. Duato. A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks:. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):841–854, 1996.

[7] I. Gazit and M. Malek. Fault tolerance capabilities in multistage network-based multicomputer systems. *IEEE Trans. Comput.*, 37(7):788–798, 1988.

[8] J-sim. http://www.j-sim.org/, May 2006.

[9] A. R. Karlin, G. Nelson, and H. Tamaki. On the fault tolerance of the butterfly. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 125–133, May 1994.

[10] T.-H. Lee and J.-J. Chou. Some directed graph theorems for testing the dynamic full access property of multistage interconnection networks. *IEEE TENCON*, 1993.

[11] T. Leighton, B. Maggs, and R. Sitaraman. On the fault tolerance of some popular bounded-degree networks. *SIAM J. Comput.*, 27(5):1303–1333, 1998.

[12] C. E. Leiserson. Fat-trees: Universal networks hardware-efficient supercomputing. *IEEE Transactions on Computers*, c-34(10), 1985.

[13] Y. Mun and H. Y. Youn. On performance evaluation of fault-tolerant multistage interconnection networks. In *SAC '92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, pages 1–10. ACM Press, 1992.

[14] S. R. Øhring, M. Ibel, S. K. Das, and M. J. Kumar. On generalised fat-trees. In *Proceedings of the 9th International Symposium on Parallel Processing*, page 39, April 1995.

[15] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, /2002.

[16] F. Petrini and M. Vanneschi. K-ary N-trees: High performance networks for massively parallel architectures. Technical Report TR-95-18, 15, 1995.

[17] http://www.quadrics.com, September 2005.

[18] F. O. Sem-Jacobsen, T. Skeie, and O. Lysne. A dynamic fault-tolerant routing scheme for fat-trees. *In Proceedings of the PDPTA conference*, 2005.

[19] F. O. Sem-Jacobsen, T. Skeie, O. Lysne, O. Tørudbakken, E. Rongved, and B. Johnsen. Siamese-twin: A dynamically fault tolerant fat tree. *Proceedings of the 19th IPDPS*, 2005.

[20] J. Sengupta and P. Bansal. Fault-tolerant routing in irregular MINs. In *TENCON '98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, volume 2, pages 638–641, December 1998.

[21] SGI. http://www.sgi.com/products/servers/altix/, September 2005.

[22] N. Sharma. Fault-tolerance of a MIN using hybrid redundancy. In *Proceedings of the 27th Annual Simulation Symposium*, pages 142–149, April 1994.

[23] M. Valerio, L. Moser, and P. Melliar-Smith. Fault-tolerant orthogonal fat-trees as interconnection networks. *Proceedings 1st International Conference on Algorithms and Architectures for Parallel Processing*, 2:749–754, 1995.

[24] M. Valerio, L. E. Moser, and P. M. Melliar-Smith. Recursively scalable fat-trees as interconnection networks. *Proceeding of 13th IEEE Annual International Phoenix Conference on Computers and Communications*, 1994.