Dynamic Fault Tolerance in Fat Trees - Research Note

Frank Olaf Sem-Jacobsen, Tor Skeie, Olav Lysne, and José Duato

***Abstract*—Fat-trees are a very common communication architecture in current large-scale parallel computers. The probability of failure in these systems increases with the number of components. We present a routing method for deterministically and adaptively routed fat-trees, applicable to both distributed and source routing, that is able to handle several concurrent faults and that transparently returns to the original routing strategy once the faulty components have recovered. The method is local and dynamic, completely masking the fault from the rest of the system. It only requires a small extra functionality in the switches to handle misrouting around a fault. The method guarantees connectedness and deadlock and livelock freedom for up to $k-1$ arbitrary simultaneous switch and/or link faults where k is half the number of ports in the switches. Our simulation experiments show a graceful degradation of performance as more faults occur. Furthermore, we demonstrate that for most fault combinations, our method will even be able to handle significantly more faults beyond the $k-1$ limit with high probability.**

I. Introduction

In order to reduce costs, most clusters, and even supercomputers, use commodity components. Now that the required computing power no longer can be achieved with a single processor, parallelisation has become the de facto method of achieving high computational efficiency and huge processing powers. Large computer systems such as supercomputers are constructed by interconnecting many smaller computing devices through a switched interconnection network. The numerous computing devices usually communicate over the interconnection network in order to cooperatively solve a large computational problem, so a critical factor of such systems is the ability of the computing devices to communicate. Many applications require large amounts of data to be transported between the computing devices. The speed at which this communication occurs will, to a large extent, dictate the efficiency of the supercomputer. Therefore, an efficient and reliable interconnection network is a crucial component of large-scale computing systems.

As the race for higher computing speeds progresses, existing technology is pushed to its limits. This also affects the interconnection network. Higher speeds leave less margin for error. Further, the increased power consumption may lead to shorter expected lifetimes of the network devices and decreased mean time between failures (MTBF). It becomes vitally important that the interconnection network is able to operate close to its nominal capacity even when network elements (typically switches and their interconnecting links) cease to function correctly.

The ability of the interconnection network to maintain a high operational efficiency, or at least to remain operational without disconnecting any computing nodes, with failing network elements depends strongly on the network topology and the routing function used to generate paths through the network. For the system to remain *connected* after a fault has occurred, there must exist a path between every pair of computing nodes that avoids the failed element.

The most simplistic fault tolerance method, and perhaps the most widely deployed, is to shut down the network that contains the faulty components and replace them. This method requires no complex mechanisms, but the time to toleration may vary from hours to days, depending on the availability of replacement hardware. However, it might not be necessary to shut down the entire system to perform the replacement.

An example of a slightly more complex mechanism is the ability of BlueGene/L to shut down the faulty parts of its network and continue functioning at reduced capacity. This example indicates another step on the path to reducing time to toleration, that of reconfiguring the network automatically either by halting it or while it remains in operation. Such methods are known as reconfiguration methods, because the network is reconfigured once the fault is discovered. A significant challenge when using reconfiguration is to guarantee that there are no dependencies between packets being routed in the network throughout the reconfiguration that may cause cycles and deadlock the network. A long-used method of ensuring this is to halt the network and its applications and drain it of all traffic before it is reconfigured, thereby guaranteeing that there is no traffic in the network during the reconfiguration. This is *static reconfiguration*. It is very time-consuming and requires that the network applications are checkpointed at regular intervals so that they may be restarted once the network is reconfigured. Much research is therefore being directed towards reconfiguring the network while it is online without introducing dependencies that may deadlock it, known as *dynamic reconfiguration*.

Reconfiguration takes time, either to run a distributed reconfiguration algorithm or to gather the information that a central entity requires to calculate the new routing tables and distribute them. Furthermore, since faulty networks must generally be viewed as irregular networks, a generic routing algorithm may be required to maintain connectivity. This approach will yield good probabilities of remaining connected despite network failures, but the use of general-purpose routing algorithms may have a severe negative effect on network performance.

The remaining approaches fall within the class of rerouting algorithms. These are inherently dynamic. By configuring the network with alternative paths from start-up it is possible to further reduce the time taken to route packets around network

faults. In the approach that we call *endpoint dynamic rerouting*, the network is configured with multiple paths between every source/destination pair. It requires much less time before the fault is tolerated than the reconfiguration methods, but some time is still required to inform the sources of the network faults when they occur. During this time, traffic already in the network will still encounter the fault and must be discarded.

The most expedient way of handling a fault is to let the devices that are directly connected to the faulty element take care of the problem. We call this approach *local dynamic rerouting*, because the fault is handled locally by the network elements. For instance, if a switch learns that one of its neighbouring switches is no longer available, it is responsible for forwarding packets on alternative paths that avoid the fault. This requires that there are preconfigured paths around all single elements that may fail in the network. This creates many possible paths in the network, and care must be taken to avoid dependencies and deadlock.

Of the approaches listed here, local dynamic rerouting has by far the lowest time to toleration and so has the lowest number of packets lost when a fault occurs. In fact, only packets that are crossing the failing element at the time of failure, or packets irreversibly buffered at the failing element, will be lost. However, due to the local nature of the paths utilised when tolerating faults, they may be suboptimal.

The large time penalty incurred by static fault tolerance may prevent the approach from providing high computational efficiency in the case of frequent fault events. This is usually the case for large supercomputers, because the probability of failure increases with the number of components. Further, the high hardware cost or long turnaround time makes it an expensive solution. Thus, in systems that have frequent faults, some of which may be transient, dynamic fault tolerance is the preferred choice. Therefore, we will adopt the local dynamic rerouting approach in this paper.

Local dynamic rerouting requires a mechanism for rerouting packets around faults locally. This can be achieved either by using an adaptive routing algorithm, or by adding a rerouting mechanism to the switches in the case of deterministic routing. We will explore both of these options.

Parallel computer systems often use the following network topologies: either direct networks such as the k-ary n-cube and mesh, or Multistage Interconnection Networks (MIN). In direct networks (e.g. mesh), all switching elements contain a computing node. On the other hand, MINs are constructed with multiple switching stages through which the packet must traverse from one processing node to another. MINs were first introduced by C. Clos in 1953 as a means of constructing nonblocking telephone switching networks [4].

A specific MIN topology is the fat-tree. The fat-tree was proposed by C. Leiserson in 1985 [9]. Most of the commercial interconnection technologies for SANs and clusters advocate the use of fat-trees or other bidirectional MINs, for instance Infiniband [1] and Quadrics [11]. Because of this, the fat-tree is to be found in many parallel computer systems. Two of the top 10 supercomputers rely on the fat-tree topology as their primary interconnection network [24] (November 2009). A classical example of a supercomputer using fat-trees is CM [13]. A more recent example is SGI Altix 3700 [29], which is used in NASA's Colombia, and the Ranger system at the Texas Advanced Computing Centre. Due to the widespread use of the fat-tree, we will constrain our proposal to this topology.

In this paper we present a dynamic local rerouting methodology for fat-trees. It can guarantee connectivity for up to and including $k-1$ arbitrary faults using either deterministic or adaptive routing, where k is half the number of ports in the switches. We implement this methodology as four different routing algorithms: link fault tolerance and link/switch fault tolerance for deterministic routing, and link fault tolerance and link/switch fault tolerance for adaptive routing. We show that using deterministic routing we require two virtual channels to guarantee deadlock freedom with more than one link fault and three virtual channels for more than one switch fault, while deadlock freedom for the adaptive version of the dynamic local rerouting algorithm does not require the use of virtual channels at all, either for link or switch faults. Furthermore, we have previously shown that the dynamic local rerouting algorithm also is applicable to source routing for link faults [16].

The rest of the paper is organised as follows. Section II introduces the k-ary n-tree, the commonly used fat-tree we consider in this paper. Section III discusses previous work relevant for dynamic fault tolerance and multistage interconnection networks. In section IV, we present a set of definitions that are needed for presenting the fault-tolerant routing mechanism in section V. We apply this mechanism to deterministic routing in Section VI, and adaptive routing in Section VII, and show that the algorithms are connected and deadlock free. In Section VIII we present a simple reconfiguration scheme for fat-trees to which we will compare our dynamic local re-routing algorithms. The algorithms are evaluated and compared in Section IX. Section X concludes the paper.

II. Background

The basic characteristic of a fat-tree is that the link capacity at every tier is constant. This is different from an ordinary tree, in which the aggregate capacity of each tier becomes smaller as we approach the tree root. The increase in link capacity at each switch tier compared to ordinary trees may be achieved by simply increasing the capacity of the links and switches in a tree with only a single root. However, the most common approach is to add additional switches and links so that the number of links and switches at every tier is the same, for instance as a k-ary n-tree [12].

A typical fat-tree with 4-port switches is depicted in Figure 1. It is a k-ary n-tree (in this case a 2-ary 5-tree) as it is defined in [12] and Definition 1. The k-ary n-tree is the class of topology we consider throughout the paper, but it is just as easy to apply the algorithms we develop here to the m-port T-tree [15].

Assume a bidirectional MIN in which the top tier is tier 0, and processing nodes are connected to the bottom tier, tier $n-1$. Every switch has k ports in each direction of the bidirectional MIN. Each of the n tiers consists of k^{n-1}

such switches. Within each switch tier, switches are numbered sequentially from zero and upwards from left to right.

The following definition states how switches should be interconnected to switches at their neighbouring tiers to form a k-ary n-tree. The definition uses a notation for n-tuples, $\{0, 1, \ldots, k-1\}^n$, which indicates that the n-tuple may be viewed as a n-digit number where each digit may have one of the values $\{0, 1, \ldots, k-1\}$. An n-tuple w may also be represented by its individual digits as $w_0, w_1, \ldots, w_{n-1}$.

Definition 1: A k-ary n-tree is composed of two types of vertices: $N = k^n$ processing nodes and $n * k^{n-1}$ communication switches, each with k ports in the upward and downward directions. Each node is an n-tuple $\{0, 1, \ldots, k-1\}^n$, while each switch is defined as an ordered pair $< w, l >$, where w is the number of the switch in the sequence from left to right represented as an n-tuple, $w \in \{0, 1, \ldots, k-1\}^{n-1}$ and $l \in \{0, 1, \ldots, n-1\}$ is the tier where it is located.

- Two switches $< w_0, w_1, \ldots, w_{n-2}, l >$ and $< w'_0, w'_1, \ldots, w'_{n-2}, l' >$ are connected by an edge if and only if $l' = l + 1$ and $w_i = w'_i$ for all $i \neq l$.
- There is an edge between the switch $< w_0, w_1, \ldots, w_{n-2}, n-1 >$ and the processing node $p_0, p_1, \ldots, p_{n-1}$ if and only if $w_i = p_i \forall i \in \{0, 1, \ldots, n-2\}$.

Note that the tier numbering is opposite of that of conventional multistage interconnection networks which usually label the tier connected to the processing nodes as tier 0. In this context upward and downward is relative to the top and bottom of the tree, i.e. upwards is towards tier 0 and downwards is towards tier $n-1$. An example of this switch labelling is shown in Figure 1.

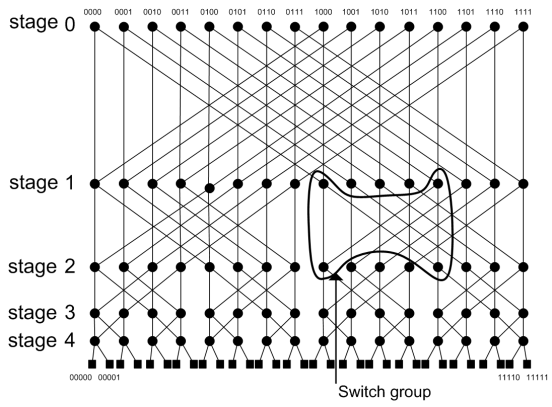


Figure 1. Switch labelling in a 2-ary 5-tree. The outline encompasses a switch group (Definition 10).

Packet routing in a fat-tree is carried out in much the same way as in an ordinary tree, or any other least common ancestor network [14]. Packets are routed towards the tree root until they reach a switch that is the least common ancestor of the source and destination, i.e., a switch that may reach both the source and destination in the downward direction, but through different links. This is called the *upwards phase*. The upwards phase is followed by a *downwards phase* in which the packet is routed downwards to its destination [1]. Given that the fat-tree has multiple roots, there will be multiple paths between any source/destination pair. This property may be utilised efficiently in a static/dynamic reconfiguration fault-tolerance scheme or an endpoint dynamic rerouting fault-tolerance scheme.

The fat-tree topology allows for fully adaptive routing in the upward phase. Every switch port that leads further up in the network is a shortest path towards a least common ancestor of the source and destination of the packet. However, in the downward phase, there is only a single shortest path from any switch to the packet destination. Hence, it is difficult to implement local dynamic fault tolerance in the downward phase, because there are no additional minimal paths that may be supplied by the algorithm for selection. As a result, it is necessary to develop a new routing algorithm that makes additional downward paths available in the case of link failure.

III. Related Work

Much of the work on fault tolerance in multistage interconnection networks focuses on providing multiple paths end-to-end to facilitate either static reconfiguration or endpoint dynamic rerouting. A frequently used approach to supply multiple paths is to add hardware to the network, in the form of additional switch stages or additional links and switches to the already existing stages [22], [28]. Another approach relies on every source being connected to more than one destination and every destination being connected to more than one source. The result is that it is impossible to isolate a given set of sources and destinations when there is a network fault; therefore, it is possible to bypass a fault using the holistic property of the network. Such a network offers *full dynamic access* and allows packets to be routed through the network in several passes by using intermediate destinations [3], [8]. Hybrid approaches have also been suggested, combining multiple paths with routing in multiple passes to achieve a greater degree of fault tolerance [21].

Similar work has been performed on types of fat-trees that do not provide multiple paths in their basic construction. An orthogonal fat-tree attempts to maximise the number of leaves connected to a tree for a given switch degree [27]. This approach supplies only a single path between any source/destination pair. To adapt the approach so that it can deal with faults, a fault-tolerant orthogonal fat-tree [28] has been proposed that supplies a number of disjoint paths between every source/destination pair by increasing the switch degree. This approach allows endpoint dynamic rerouting or static reconfiguration. However, it may not be used to achieve local dynamic rerouting in a straightforward manner.

A comprehensive study of the fault-tolerant properties of several multistage interconnection networks is performed in [10]. None of the methods evaluated consider dynamic fault tolerance, and the authors found that enhancing MIN topologies with additional hardware may lead to a decrease in performance compared to the original network.

[1]Note that relative to the enumeration given in Definition 1, a packet in an upwards phase will traverse decreasing switch tier numbers, and a packet in the downward phase will traverse increasing tier numbers.

To provide local dynamic rerouting, Sengupta et al. [20] propose a modified version of the single path Omega network to create multiple paths between the source/destination pairs, which may be used to avoid faults dynamically while the packet is traversing the network. Furthermore, Sengupta and Bansal [19] propose another topology, called the Quad Tree, which consists of two parallel Double Trees. There are links interconnecting the same stages in the two trees, which allow packets to be re-routed dynamically. Both topologies are only able to tolerate a single fault in any stage. Yet another network topology that supports dynamic rerouting is the modified MIN presented in [26]. Redundant links are placed between switches at the same stage. This allows a packet to be forwarded to another switch at the same stage when a fault is encountered. The same approach is used in [18], where crossover links between two parallel fat-trees are used for local dynamic fault tolerance. This method may, in principle, be applied to many MIN topologies, but it still only provides tolerance of a single fault.

FRoots [23] is a topology-agnostic routing algorithm that is designed to provide both end-to-end and local dynamic fault tolerance. It is based on the idea that every network node should be a leaf of a tree, so that if a node fails, no traffic passes through it other than traffic to and from it. This is achieved by constructing several Up*/Down* routing graphs such that each node is a leaf in at least one graph. Each Up*/Down* graph is implemented in its own virtual layer. When a packet encounters a faulty network element, it may be switched to a new layer with a graph in which the faulty elements are leaves. It will then be forwarded to its destination along a different path.

Imposing tree topologies on the fat-tree so that every switch and processing node in the network is a leaf in one of the trees is difficult, because the fat-tree is already a tree topology. A large number of Up*/Down* graphs, and thus virtual layers, will be required to ensure this. In addition, the complexity would lead to a much longer path when a fault is encountered. Clearly, such a generic solution is suboptimal when considering a regular topology such as the fat-tree. It requires a large number of virtual channels to form the necessary virtual layers and the path length will be longer, even when the network is fault-free.

To the best of our knowledge, there exists no previous method that provides local dynamic rerouting for several simultaneous network faults without adding additional hardware and without requirement of additional virtual channels. We argue that with the sizes of the supercomputers that are currently being built, local dynamic fault tolerance is very appealing, if not mandatory.

IV. Definitions

In this section, we present a set of definitions that provide the necessary framework to show that the fault-tolerant routing algorithms that we will present in the next sections are actually fault-tolerant and that they are deadlock free.

We begin by defining adaptive and deterministic routing algorithms in general.

Definition 2: An *adaptive routing function* R* supplies a set of possible output queues to be used by an incoming packet on input port i on switch s to reach a destination d. When a set of possible output queues for a packet has been given by a routing function, the actual queue into which the packet is forwarded is chosen by a *selection function*.

Definition 3: A *deterministic routing function* R* supplies one output queue to be used by an incoming packet on input port i on switch s to reach a destination d. The output queue is the same for all incomming packets to the same destination.

Note that the incoming port i and the associated virtual channel may or may not be part of the routing function, depending on what is required by the algorithm and supported by the hardware.

Definition 4: A *link* is a connection between two switches that are connected by an edge (as defined in Definition 1), allowing information such as data packets to pass between them.

A downward link from a switch at tier l is the connection to a switch at tier $l+1$. Recall that we assume that queues are located at the switch outputs. A downward queue is therefore a queue at a switch output port that is connected to a switch one tier lower down (from tier l to $l+1$) by a link. Similarly, an upward queue is a queue at a switch output port that is connected to a switch one tier higher up by a link (from tier $l+1$ to l). This combination of queue and link may also be called a *channel*.

Let us next define the faults we consider for our routing algorithms.

Definition 5: A *link fault* is the total benign permanent/transient failure of a link or either of the switch ports connecting a switch to the link.

Definition 6: A *switch fault* is the total benign permanent/transient failure of a switch.

The following definitions help us to name parts of the fat tree.

Definition 7: The *least common ancestors* (lca) of a source s and destination d are all the switches at tier l where $l = min(j), s_j \neq d_j$, and $lca_i = s_i = d_i \forall i < l$

Recall that the routing in the fat-tree consists of two phases: (i) the upward phase from the source up the tree towards one of the least common ancestors (towards tier 0), and then (ii) the downward phase from the least common ancestor down the tree (through increasing tier numbers) to the destination. The length of the path followed by the packet is determined by the tier at which the least common ancestors of the source and destination are located.

To demonstrate the fault-tolerance capabilities of the routing algorithm that we will present, we must know which part of the network is affected by the fault in that it carries the additional load of the packet that has to be routed around the fault.

Definition 8: Two switches are *neighbours* if there is a link connecting them to each other.

The switches can only be neighbours if they are at neighbouring switch tiers l and $l+1$.

Definition 9: 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).

Two such interconnected sets make up an integral part of the dynamic local routing algorithm we will present later. To identify the possible paths taken by a packet that encounters link faults, we must know something about the relationship between the switches in different tiers of the fat-tree. This is expressed in the next definition.

Definition 10: A *switch group* g in a k-ary n-tree is the union of two completely interconnected sets of switches, a and b, where a contains only switches at tier l and b contains only switches at tier $l+1$. a contains all neighbours at tier l of all switches in b, and b contains all neighbours at tier $l+1$ of all switches in a, $g = a \cup b$.

And now we describe the same switch group in the context of the k-ary n-tree definition.

Definition 11: A *switch group* G in a k-ary n-tree is made up of all switches at the switch tiers l and $l+1$ where $v_i = u_i \forall i \neq l$, and v,u are n-tuples as defined in Definition 1.
More informally, if each switch has k links in the upward direction, a switch group consists of $2k$ switches. Of these, k switches are at the lower tier of the group, tier $l+1$, and k are at the upper tier, tier l. Every switch at the lower tier of the group has a link that connects it to all switches in the upper tier and vice versa. An example of a switch group is illustrated in Figure 2.

The concept of the switch group can be extended to encompass a two-hop switch group, a switch group that consists of three tiers of switches instead of two. This is necessary to tolerate switch faults.

Definition 12: A *2-hop switch group* G_2 is made up of all switches at three neighboring switch tiers $l, l+1, l+2$ where $v_i = u_i \forall i \neq j, j = [l, l+1]$, and v,u are n-tuples as defined in Definition 1.

Routing in the fat-tree prohibits packet transitions between certain queues (downward to upward) in order to guarantee deadlock freedom, a state in which it is impossible for the network to experience deadlock. However, when faults are introduced into the network, some of the illegal packet transitions must be performed in order to keep all processors connected. The next definitions identify where in the fat-tree these illegal transitions take place.

Definition 13: A *U-turn* is a part of a packet's path where a downward queue (from tier l to tier $l+1$) is followed by an upward queue (from tier $l+1$ to tier l).
Thus, the term 'U-turn' does not apply to the upward to downward transition that separates the upward and downward paths in the original routing algorithm.

Definition 14: A *U-turn switch* is the switch that contains the upward queue of a U-turn.

The next set of definitions will be used when considering the deadlock freedom and connectivity of the fault-tolerant algorithm that we will propose.

Definition 15: A network is *deadlocked* if there exists a set of full queues Q such that all queues supplied by the routing function for the first packet in every queue in Q are also in Q.

More formally, there are many conceivable assignments of packets to queues in a network, but only some of these assignments are actually possible, given a particular routing function.

Definition 16: A *legal configuration* for the routing function R* is an assignment of packets to network queues that may be reached from an empty network following R*.

Definition 17: A configuration is *deadlocked* if the configuration is legal and there exists a set of queues Q such that all possible next-hop queues for the first packet in any queue $\in Q$ is also $\in Q$.

Definition 18: A routing function R* is *connected* if it establishes a path to the destination for every packet in every legal configuration.

V. Dynamic Local Rerouting

We will start by giving the reasoning behind the dynamic local re-routing mechanism that we propose. The basic mechanism is the same, regardless of whether we consider adaptive or deterministic routing and link or switch faults. We show in this section that when using our proposed re-routing mechanism the network remains connected for $k-1$ arbitrary faults, and is livelock and deadlock free for one arbitrary fault, regardless of whether adaptive or deterministic routing is utilised. We show in the following sections how the re-routing mechanism can be guaranteed livelock and deadlock free for $k-1$ arbitrary faults using both deterministic and adaptive routing. However, the actual implementations of the algorithms and the proofs to achieve this differ for the two routing schemes, so they will be discussed separately.

The fault model we consider is that of permanent or transient benign faults that totally disables single ports or entire switches in the networks. Tolerating the failure of the links and switches that connect processors to the fat-tree will require multiple network interface cards for each processing node, and each of these nodes must be connected to multiple access points in the fat tree. This case must be solved by introducing additional hardware, rather than by re-routing, as was done with the Siamese-twin fat-tree topology [18].

Given an arbitrary source and destination, it follows from Definition 1 that every possible upward queue (from tier $l+1$ to tier l) in the upward phase of the routing algorithm leads towards a least common ancestor. Fault tolerance in the upward phase of the routing algorithm is therefore quite simple to achieve. Whenever the output supplied by the routing algorithm is faulty, a re-routing mechanism or adaptivety provides one of the other upward ports, which we know leads to a least common ancestor.

The downward phase is not as straightforward, because there are no alternative shortest paths from a switch connected to a faulty downward link /switch. The packet must therefore be re-routed on a longer path.

A. Link Faults

It can be seen from the fat-tree topology and Definition 10 that every switch is part of one or two one-hop switch groups, one in each direction in which it has neighbours. The group consists of (i) the switch itself and $k-1$ other switches at the same tier, and (ii) k switches at one of the two neighbouring

tiers (definition 10). In other words, every switch in a fat-tree except a bottom-tier switch (tier $n-1$) is an upper-tier switch in a switch group. Similarly, every switch in a fat-tree except a top-tier switch (tier 0) is a lower-tier switch (larger l) in a switch group.

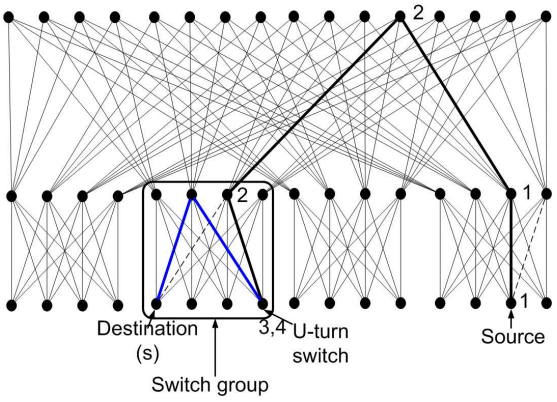


Figure 2. A fat-tree consisting of radix 8 switches with two link faults. The faulty links are marked as dotted lines, and the bold line describes the path of a packet from its source to its destination. Note how the packet is re-routed within the group via the U-turn switch. The numbers refer to the corresponding steps of the routing algorithm.

When a packet encounters a faulty link in the downward phase, the path to the destination that involves only downward movement is disconnected. We must therefore expand the routing algorithm in the downward phase to make a larger number of paths available. Let s denote the next hop switch that is unreachable from c because of the link fault (Figure 2). The switch in which the packet currently resides (c) is an upper-tier switch of a switch group, and s must therefore be a lower-tier switch in the same group. We now propose to reach s by re-routing the packet within the group; first to an arbitrary lower-tier switch in the group, and from there to an arbitrary upper-tier switch in the group (different from c). It follows from the definition of a group that there should be a link from this arbitrary upper-tier switch (marked "u" in Figure 2) down to s. The lower-tier switch to which the packet is re-routed becomes a U-turn switch, because a downward to upward transition takes place (Definition 14). A U-turn switch recognises a re-routed packet as a packet arriving from an upward port for which it only has upward output ports in the forwarding table, i.e. the packet is in the downward phase, but has no downward path from the switch.

Lemma 1 proves that there are k disjoint paths between any two switches at the upper tier of a one hop switch group; i.e. between c and any other switch at the same tier within the group.

Lemma 1: A switch u at the upper tier l in a 1-hop group G_1 at switch tiers $l, l+1$ can reach any other switch v at the same tier in G_1 with a path length of 2 hops with fewer than $k-1$ link faults in in the group.

Proof: Any switch c at the upper tier (l) of a switch group has k neighbours at the lower tier ($l+1$). Each of these neighbours are neighbours to all switches in the group at the same tier as c. Thus, there exist k paths from c to each of the other switches in the group of the same tier, one through each of its neighbours in the group. There are no links between switches at the same tier; hence, these paths will be the shortest paths, each with a length of two hops. $k-1$ link faults are not sufficient to disconnect k disjoint paths, so there exists a path of length two hops within the group between any two switches at the upper tier of the group. ■

Using this lemma we can show that there exists a path from any healthy switch in the network to any destination given $k-1$ arbitrary link faults when using the 1-hop re-routing mechanism presented above.

Lemma 2: Every switch has a path to all destinations with fewer than k arbitrary link faults when re-routing down one tier on encountering link faults in the downward phase.

Proof: For all destinations reachable upwards from any switch there are k links providing this connectivity. With $k-1$ link faults one of these links will always be available, providing a path to move the packet one hop closer to the destination. For destinations reachable in the downward direction from a switch we have the following argument: When there are $k-1$ link faults in the system, at least one of the k upper tier switches in a group with link faults will not be connected to any faulty link. We call this switch S_t. Let the upper tier switches of the switch group be at tier l, then the lower tier switches of the same group are at tier $l+1$. From Lemma 1, there are k disjoint paths between any two switches at the upper tier in a group. $k-1$ link faults are not enough to disconnect all these paths, and thus, any upper tier switch connected to a faulty link is able to reach S_t, which we know has a healthy link that moves the packet one hop closer to its destination. Once the packet has reached a lower tier switch (tier $l+1$) in the group with a valid downward path, subsequent link failures encountered will be tolerated in a different group, further down in the tree. ■

B. Switch Faults

For switch-fault tolerance, re-routing down one tier is not sufficient to avoid the faulty switch, as all paths to a specific destination d within the switch group will lead through the same switch s. However, re-routing down two tiers instead of just one will avoid the faulty switch s and achieve connectivity. In this case we say that the faulty switch s is located at the middle tier of a two-hop switch group G_2 (Figure 3). We do not need the concept of switch groups to tolerate the failure of the top tier switches, as the failure of these need only be considered in the upward routing phase. Note that re-routing down two tiers to tolerate switch fault also implies toleration of link faults. However, for link faults between the two bottom tiers of the fat-tree it is obviously not possible to re-route down two tiers for packets located at tier $n-2$. Therefore, to guarantee full link fault tolerance, as well a switch fault tolerance, it is necessary for the bottom tier switches to act as U-turn switches even though the re-routing mechanism is configured for two hops. Also, recall that extra hardware is required to tolerate the failure of the bottom-tier switches. In order to keep track of the number of tiers the packet has been re-routed down an extra field in the packet header, which we call the *port* field, is required. This field is initially set to -1 to

indicate that the packet is not re-routed. After re-routing down the first tier (from l to $l+1$), it is set to -2 to inform the next receiving switch that it is the U-turn switch. Upon forwarding the packet upwards (from $l+2$ to $l+1$), the receiving switch at the tier above the U-turn switch (tier $l+1$) records the incoming port number in the field in the packet header. This allows a packet that is returning to tier l and that encounters another switch fault to return to the same U-turn switch from which it arrived. It will first be forwarded down the same link to tier $l+1$ from which it arrived, and then, using the *port* field in the packet header, it can be returned down to the u-turn switch at tier $l+2$. All U-turns performed by that packet to avoid switch faults at tier $l+1$ will be through the one U-turn switch. Lemma 3 demonstrates the same property for two-hop switch groups as Lemma 1 did for one-hop switch groups.

Lemma 3: A non-faulty switch u at the upper tier l in a 2-hop group G_2 over switch tiers $l, l+1, l+2$ can reach any other non-faulty switch v at the same tier in G_2 with a path length of 4 hops if there are fewer than k switch and link faults in the group.

Proof: The middle-tier switches in the two-hop switch group are at switch tier $l+1$ in the fat-tree, and they consist of all possible permutations of the l'th and $(l+1)$'th digits in their n-tuples, with all other digits being equal. A digit has k possible permutations. Thus, when there are $k-1$ switch faults in the group, there exist k switches in the switch group at tier $l+1$, each with the same permutation of their l'th digit, and a unique permutation of their $(l+1)$'th digit that are fault free. We call this set of switches F. Thus, following from Definition 1, there is at least one switch T at the lower tier in G_2 that is fault free and connected to all switches in F. T is consequently able to reach any non-faulty upper-tier switch in G_2 through a switch in F. Any non-faulty upper-tier switch in G_2 can thus reach any other non-faulty upper-tier switch in G_2 through T.

The case for link faults is identical. As every switch fault is associated with $2 * k$ failed links, the necessary paths are still available with $k-1$ link faults. ■

We now continue with proving the same degree of connectivity for $k-1$ switch faults assuming we re-route down two tiers when encountering switch faults in the downward phase.

Lemma 4: Every switch has a path to all destinations with fewer than k arbitrary link or switch faults (assuming no faults in the bottom switch tier) when re-routing down two tiers on encountering switch faults in the downward phase.

Proof: For all destinations reachable upwards from any switch there are k links to k different switches providing this connectivity. With $k-1$ link or switch faults one of these links will always be healthy and connected to a healthy switch, providing a path to move the packet one hop closer to the destination. For destinations reachable in the downward direction from a switch we have the following argument: When there are $k-1$ link or switch faults in the system, at least one of the k^2 upper tier switches in a two-hop switch group with $k-1$ switch faults at the middle and lower tier will not be connected to any faulty switch or link. We call this switch S_t. Let the upper tier switches of the switch group be at tier l, then the lower tier switches of the same group are at tier $l+2$. From Lemma 3, any upper tier switch connected to a faulty switch is able to reach S_t, which we know is connected to a healthy switch at tier $l+1$ with no faulty links, one hop closer on the path to the destination. Once the packet has reached a switch on the shortest path to the destianation at a lower tier ($> l$), it enters a new group, so subsequent switch failures encountered will be tolerated in a different group, further down in the tree.

With at most $k-1$ link faults between the two bottom tiers of the fat-tree, link fault tolerance is guaranteed since this reduces to rerouting in a one hop switch group around the link fault(s). ■

C. The general algorithm

A generic version of the algorithm for link faults is given below. The numbers of the points in the algorithm respond to the numbers in Figure 2.

1) In the upward phase towards the root, a link is provided by the forwarding table as the packet's outgoing link. If the link is faulty, it is disregarded and one of the other upward links are chosen by the re-routing mechanism, re-routing the packet.
2) In the downward phase (towards tier $n-1$), only a single link is provided by the forwarding table; namely, the link on the shortest path from the current switch to the destination. If this link is faulty, the following actions are performed:
   a) If the packet came from the switch tier above (from l to $l+1$), another arbitrary downward link is selected by the re-routing mechanism, which forces the packet to be re-routed. This will result in the packet being forwarded as long as there are fewer than k link faults.
   b) If the packet came from the switch tier below (from $l+1$ to l), the packet is re-routed back down to the same lower-tier ($l+1$) switch again. This is necessary to avoid livelock.
3) A switch that receives a packet from a link connected to an upper tier for which it has no downward path is a U-turn switch.
4) The U-turn switch chooses a different upward port through which to forward the packet that has not been previously tested.
5) If all upward ports from the U-turn switch have been tested, the path is disconnected and the packet must be discarded.

Which output port is chosen from the U-turn switch in point 4 and how the switch knows that all upward ports have been tested in point 4 and 5 is specific to the deterministic and adaptive routing schemes and will be detailed in the next sections. We have shown that this algorithm is connected and [17] (or Appendix A) shows that it is deadlock and livelock free with one link or switch fault without the use of additional resources.

VI. Deterministic Routing

We have shown that the re-routing mechanism presented in the previous section is sufficient to guarantee a connected path

from every switch to every destination. However, we have yet to consider how we can be guaranteed to find this path in the presence of $k-1$ faults. We will now give a deterministic routing algorithm which is able to find the connected path while remaining livelock and deadlock free.

The basic algorithm consists of regular upward-downward fat-tree routing combined with re-routing in the downward phase around the link faults. To ensure that the routing algorithm is deadlock-free, a strict ordering must be imposed on the sequence of upward links (ports) to test from a U-turn switch when a packet is being re-routed. In the rest of this section, we call this sequence D.

A. Link Faults

$k-1$ link fault tolerance requires the use of an additional virtual layer, the re-routing layer, in which packets will be forwarded while they are being re-routed. A virtual layer, VL, is a set of virtual channels such that every bidirectional link has a bidirectional virtual channel (VC) in the virtual layer. For simplicity, we assume that VC1 on all links belongs to VL1, the normal layer (NL), and VC2 on all links belongs to VL2, the re-routing layer (ML).

Based on this, the deterministic dynamic fault-tolerant algorithm, $R^{link}_{deterministic}$, for tolerating up to and including $k-1$ link faults is presented below as additions to the generic algorithm presented in the previous section. The steps of this algorithm are marked in Figure 2 by numbers corresponding to the steps in the algorithm. $R^{link}_{deterministic}$

1) The packet is forwarded in NL.
2) ...:
   a) The packet is forwarded in NL.
   b) The packet is forwarded in ML.
3) All subsequent forwarding of this packet through this U-turn switch will take place in ML.
   a) If the packet arrives in NL the U-turn switch chooses the first upward link in testing sequence D.
   b) The packet arrives in ML the U-turn switch chooses the next upward link in the testing sequence D after the port through which the packet arrived and subsequently forwards the packet up this link in ML. This is always possible with $k-1$ faults since D covers all the k upward ports of the U-turn switch.
4) A U-turn switch that receives a packet from an upward link that is the last link in the sequence D discards the packet.

After reaching point 5 of the algorithm, the switch may inform the source node of the failure to prevent it from transmitting more packets. This can only happen when the network contains more than $k-1$ link faults.

Local dynamic re-routing one-link fault tolerance can be achieved with a simpler version of the algorithm, given that we know that any path taken by the packet after the U-turn switch is fault free. Points 4 and 5 in $R^{link}_{deterministic}$ can therefore be ignored in this case. An arbitrary upward link can be used from the U-turn switch in point 3; there is no need for a test sequence. In addition, there is no need for additional virtual channels, because a deadlock requires at least two U-turns in the network as we proved in [17] (or Appendix A). We can guarantee the existence of only one U-turn by deterministically selecting the same downward re-routing link on encountering the link fault. All re-routed packets will always reach the same U-turn switch through the same downward link, and thus there is only one U-turn. Livelock freedom and $k-1$ link fault tolerance for $R^{link}_{deterministic}$ has previously been proved in [17] (or Appendix A), and deadlock freedom will be shown in the next section.

B. Switch Faults

In order to tolerate switch faults we need to modify the packet header to include the port field as we have indicated previously. This is required to be able to efficiently re-route down two tiers. Furthermore, to guarantee that this will remain deadlock free we require three virtual channels/layers, one more than for link faults. The deterministic dynamic switch fault-tolerant algorithm, $R^{switch}_{deterministic}$, for tolerating up to and including $k-1$ switch faults is presented below. The steps of this algorithm are marked in Figure 3 by numbers corresponding to the steps in the algorithm.

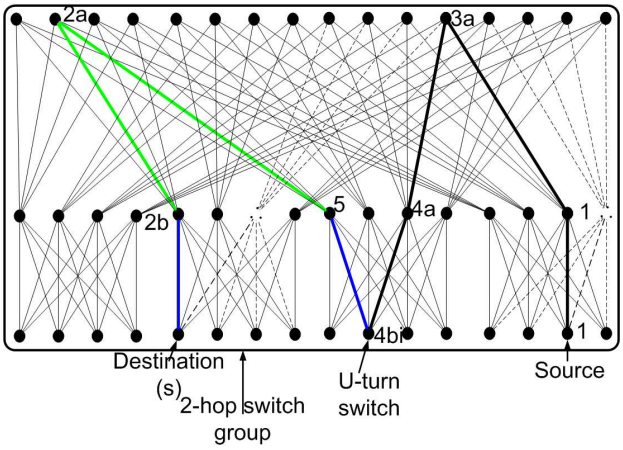


Figure 3. A fat-tree consisting of radix 8 switches with two switch faults. The faulty switches and their links are marked as dotted lines, and the bold line describes the path of a packet from its source to its destination. Note how the packet is re-routed within the group via the U-turn switch. The numbers refer to the corresponding steps of the routing algorithm.

$R^{switch}_{deterministic}$

1) In the upward phase towards the root, one link is provided by the forwarding table as the packet's outgoing link. If the switch connected to the link is faulty, it is disregarded and one of the other upward links are chosen by the re-routing mechanism, re-routing the packet. The packet is forwarded in NL.
2) In the downward phase (towards tier $n-1$), only a single link is provided by the forwarding table; namely, the link on the shortest path from the current switch to the destination.

a) If the packet arrives from below in ML2, forward the packet downwards in ML2.
b) If the packet arrives from above in ML2, forward the packet downwards in ML1.
c) If the packet arrives in ML1, forward the packet downwards in NL.
d) If the packet arrives in NL, forward the packet downwards in NL.

3) If the switch connected to this downward link is faulty, the following actions are performed:
   a) If the packet came from the switch tier above (from l to $l+1$), another arbitrary downward link is selected, which forces the packet to be re-routed. This will result in the packet being forwarded as long as there are fewer than k link faults.
      i) If the packet is in ML2, forwarded in ML1.
      ii) If the packet is in ML1 or NL, forward it in NL.
   b) If the packet came from the switch tier below (from $l+1$ to l), the packet is re-routed back down to the same lower-tier ($l+1$) switch again. This is necessary to avoid livelock. The packet is forwarded in ML2.
4) A switch that receives a packet from a port connected to a switch at the tier above, for which it has no downward path takes the following actions:
   a) If the *port*-field equals -1 another arbitrary downward link is selected, which forces the packet to be re-routed. This will result in the packet being forwarded as long as there are fewer than k switch faults. The packet is forwarded in NL.
   b) If the *port*-field equals -2 or the *port*-field equals -1 and the switch is a bottom-tier switch (at tier $n-1$) this switch is a U-turn switch.
      i) If the packet is in the normal channel, forward the packet upwards through the first port specified by the re-routing sequence D from which the packet did not arrive. As long as there are fewer than k switch faults one of these ports will be connected to a fault free switch.
      ii) If the packet is in ML1, forward the packet through an upward output port in ML1 following the incoming port in the re-routing sequence D.
      iii) If the incoming port is the final output in the re-routing sequence D and the incoming layer is ML1, discard the packet. All upward links of the U-turn switch have been tested and there is no available path to the destination from this switch. This may only happen when there are k or more faults.
   c) If the *port*-field does not equal either -1 or -2, forward the packet down through the port indicated by the *port*-field and set the *port*-field to -2. The packet is forwarded in ML1.
5) A switch that receives a packet from a port connected to the lower tier in which the *port*-field equals -2 stores the incoming port number in the *port*-field and forwards the packet in ML2 through one of its upward links. The algorithm is then repeated from step 2.

After reaching point 4.b.iii) of the algorithm, the switch may inform the source node of the failure to prevent it from transmitting more packets. This may only happen when the network contains more than $k-1$ link and switch faults.

If we consider only one switch fault, it is sufficient that this field only contains one bit to indicate whether the packet is re-routing down the first or the second tier (from l to $l+1$, or from $l+1$ to $l+2$). Furthermore, any packet will be re-routed only once before it reaches its destination; hence, there can be no livelock. Thus, no modifications are required in order to ensure freedom from livelock apart from the re-routing bit in the packet header, not even a re-route vector in the switches.

Just as for link faults, $R^{switch}_{deterministic}$ allows transparent re-utilisation of previously failed elements. The proof of connectivity and livelock freedom is available in [17] (or Appendix A).

C. Deadlock Freedom

In this section we show that $R^{link}_{deterministic}$ is deadlock free. The method for showing that $R^{switch}_{deterministic}$ is deadlock free is similar, except that three virtual layers are required. This proof is available in [17] (or Appendix A). As we have indicated, to guarantee that $R^{link}_{deterministic}$ is deadlock free with more than one fault, we require the use of two virtual layers: (i) a normal virtual layer (NL) where most of the packet forwarding takes place, and (ii) a re-routing layer (ML) that contains the packets currently being re-routed. The transition from NL to ML takes place in the U-turn switches. Each time a packet performs a U-turn it enters ML. The transition from ML back to NL takes place in those lower-tier switches in the switch group that are on the packet's path to the destination. In other words, the transition back to NL takes place when the re-routed packet has reached the switch at the bottom end of the failed link around which the packet was initially re-routed.

An obvious property of a cyclic dependency chain is that if each channel in the cycle were to be assigned a number larger than the previous channel in the cycle, there would, sooner or later, be a dependency from a channel with a high sequence number to a channel with a lower sequence number. Consequently, if it is possible to apply a set of numbers to all channels in the network in such a way that any packet traversing the network always moves to a channel with a higher sequence number than the previous channel, there can never be any cycle.

In order to show that $R^{link}_{deterministic}$ is deadlock-free we introduce a numbering scheme which we apply to all channels in the fat tree. The fat-tree consists of four types of channels: upward and downward channels in NL, and upward and downward channels in ML. Let N^u_s be the set of upward channels in NL at link tier s and N^d_s the set of downward channels in NL at link tier s. M_s is the set of upward and downward channels at tier s in ML, and M^u_s and M^d_s are the subsets of the upwards and downwards channels respectively. u' is the sequence number assigned to the channels incident

of that all years five and it u. The assignment $u' = j$ where j is an integer greater than -1 and $u \in N \cup M$ assigns the number j to the channel u. The number of link tiers in the network is $n-1$, and s is the link-tier number ($[0, n-2]$). The tier number of a link is same as the tier number of the switch connected to its upward end. Hence, the link-tier number of the links connected to the root switches as tier 0 is also 0, and the link-tier number of the links connecting the leaf switches at tier $n-1$ to the swithces at tier $n-2$ is $n-2$. We number the channels as follows:

1) $u' = n-2-s, \forall u \in N_s^u, s \in \{0 \dots n-2\}$
   All upward channels in NL are assigned the reversed indexing of the link tiers.
2) $u' = v' + 1, \forall u \in N_0^d, \forall v \in N_0^u$.
   All downward channels in NL at the topmost link tier are assigned a number that is one unit greater than the topmost upward channels in NL.
3) $u' > v', \forall u \in N_s^d, \forall v \in M_{s-1} \cup N_{s-1}^d, s > 0$.
   Every downward channel at tier s in NL is given a number that is larger than any downward channel in NL and any channel in ML of all tiers l higher than s ($l < s$).
4) $u' = v' + 2i + 1, w' = v' + 2i + 2, \forall u \in D \in M_s^u, w \in D \in M_s^d$, i is the index of u in the sequence D, $i \in \{0, \dots, k-1\}$, $\forall v \in N_s^d$.
   Every re-routing channel is given a number larger than the normal downward channels at tier s, in an increasing order from $v'+1$ corresponding to the index of the re-routing channel in the re-routing sequence D from the u-turn switch. For instance, assuming that the largest sequence number of all downward channels in NL at tier s is 10, the first upward re-routing channel in the sequence to be tested from a U-turn switch at tier s is 11, the first downward re-routing channel is 12, the second upward re-routing channel is 13, and so forth.

We now show that the proposed numbering scheme provides all channels in the network with a single number.

Lemma 5: Every channel in the fat-tree can be assigned one and only one number if that satisfies all four points of the numbering algorithm.

Proof: We consider first the channels in NL. Every channel in the fat-tree belongs to a single link tier. Rule 1 guarantees that all upward channels in NL are assigned a number. The only requirements on these numbers is that they be the reverse order of the link tier indexing, which is trivial to guarantee. There is only one possible assignment for any link.

For the downward channels in NL, rules 2 and 3 guarantee that every channel is assigned a number. All downward channels at the topmost link tier only have one possible assignment as per rule 2. From point 4, the maximum number assigned to the re-routing channels at a tier l is limited by the number of channels in the re-routing sequence, which is $2 * k$. The maximum number assigned to any channel at a tier above tier l is $n-2+l+l*2k$ (the numbers assigned to all channels in N^u, plus the number assigned to the downward channels at each tier, plus all the numbers assigned to re-routing channels at each tier above l). Any downward channel at tier l in NL may therefore be assigned a number that satisfies rules 2 and 3, which is one greater than $n-2+l+l*2k$. There is obviously only one such number for any link at any link stage.

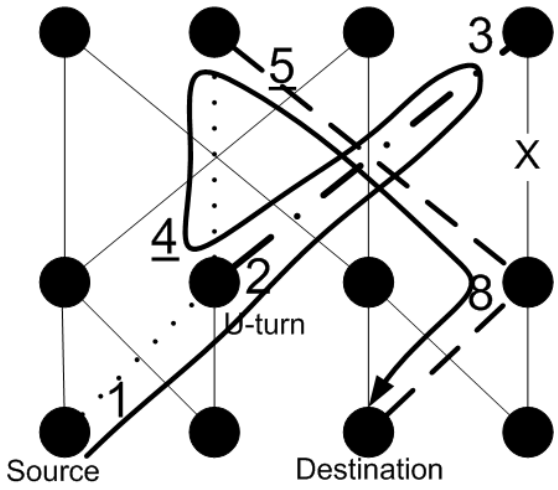


Figure 5. The path followed by packets from source to destination. The link marked "X" is faulty, and since this is on the deterministic path, the packet must be re-routed. The dotted links carry packets of the flow upwards in the network, dashed links carry packets downwards, and the dashed and dotted link carries packets of the flow both ways. The numbers indicate the number of the channel that the packet occupies out from the switch according to our numbering scheme, the underlined numbers being in the re-route layer. The packet path is indicated by the arrowed line.

Similarly, rule 4 guarantees that all re-routing channels at all tiers are assigned a number. Rule 2 guarantees that all top-tier downward channels in NL are assigned the same number n. Assigning the number $n+i$ to consecutive channels with index i in the re-routing sequence D, combined with the fact that any re-routing channel belongs to the re-routing sequence of one and only one U-turn switch, guarantees that we conform to rule 4 for the top-tier re-routing channels and that every re-routing channel is assigned exactly one number. For re-routing channels at subsequent lower link tiers, the same reasoning applies, except that the number of the downward channels in NL of each tier l is given by $l * 2k + l + n - 2$, as shown previously. ■

This numbering scheme is illustrated in Figure 4 for a small 2-ary 3-tree. There is one figure for each of the upward and downward channels in the normal and re-routing layers. Figure 5 illustrates the path followed by a flow of packets from the node marked 'source' to the node marked 'destination'. The dotted links carry packets of the flow upwards in the network, dashed links carry packets downwards, and the dashed and dotted link carries packets of the flow both ways. The link marked "x" is faulty, and since this is on the deterministic path the packet must be re-routed. Since $k = 2$, packets must therefore backtrack down one tier towards the source to the U-turn switch, as illustrated by the dashed and dotted line. The numbers indicate the number of the channel that the packet occupies out from the switch, according to our numbering scheme. It is clear that the re-routed packets only cross channels with increasing numbers, thereby guaranteeing that the routing algorithm will never deadlock.

Before we proceed with the formal proof of deadlock

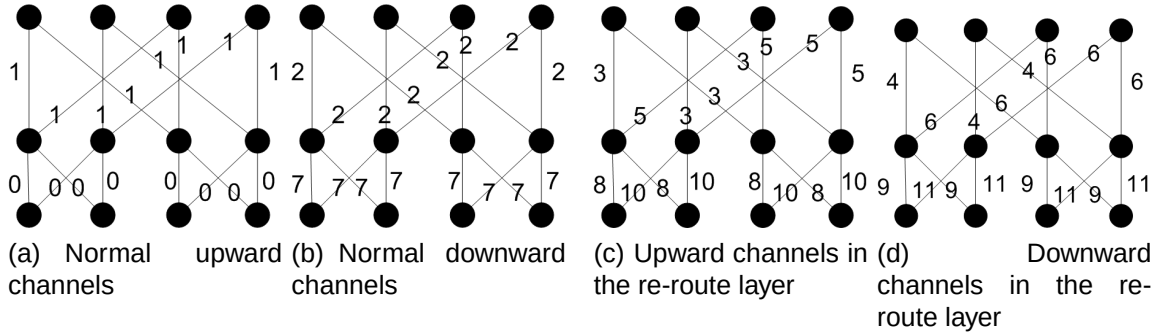


Figure 4. Channel numbering in a fat-tree for link fault tolerance. Numbering of the different channels in the normal and re-routing layers, assuming that the ordered sequence D tests the upward ports from left to right.

freedom, we must consider one final corollary, which shows that any upper-tier switch in a switch group will have the same index in the re-routing sequence D for any U-turn switch in that group. This means that all upward channels in the re-routing layer to a given switch will have the same sequence number i, and all downward channels in ML from this switch will have the same sequence number $i+1$ (This follows from rule 4).

Corollary 1: All upward links to an upper-tier switch u in a switch group G have the same index i in the re-routing sequence D of all U-turn switches in G.

Proof: Consider a switch group G with its upper tier at tier l, and its lower tier at tier $l+1$. From Definition 1, all switches in G are connected to port $w_l + k$ of the switches at tier $l+1$, where w_l is the (l)'th tuple of the switch n-tuple. Any upper-tier switch is therefore connected to the same port number on all the lower-tier switches, and these will have the same index in the re-routing sequence D for all possible u-turn switches in the group.

■

We proceed with the proof of deadlock freedom for the routing algorithm. We consider here the dependencies between channels in the network. Since every channel in the network has its own number, it is sufficient to show that the next hop channel for any packet in any node in the network has a number that is larger than the sequence number of all channels that the packet may previously have traversed. When following such a dependency chain where all channels have a number, all next-hop channels that could possibly close a cycle will necessarily be assigned a number that is lower than or equal to the current channel.

Theorem 1: $R^{link}_{deterministic}$ is deadlock-free.

Proof: We give all channels in the network a sequence number conforming to the above rules. The routing algorithm is deadlock-free if we can show that it is impossible to move to a channel that has a sequence number lower than or equal to the current channel.

There are four specific situations for any packet forwarded in the network.

1) *The packet is forwarded upwards in NL*
   According to rule 1, all upward channels at the next link tier upwards have a larger sequence number than the current channel.
2) *The packet is forwarded downwards in NL*
   According to rules 2 and 3, all downward channels at the next link tier downwards have a larger sequence number than the current channel, regardless of whether the current channel is a normal channel or a re-routing channel.
3) *The packet is forwarded upwards in ML*
   This only takes place in the downward routing phase. According to rule 4, all re-routing channels at the current link tier have a larger sequence number than any downward normal channel at the same tier, and all re-routing channels at the tiers above. In addition, the next re-routing channel always has a larger sequence number than the current re-routing channel, following the ordered sequence D and rule 4.
4) *The packet is forwarded downwards in ML*
   There are two possibilities. Either the packet must return to the U-turn switch, or it has a fault-free downward path towards the destination. We call the current switch z. z has the same position in the re-routing sequence D for all U-turn switches in the switch group. Thus all upward re-routing channels in the switch group connected to z will have the same sequence number, as will all the downward re-routing channels in the switch group connected to z (rule 4, see Figure 4 for an example). The sequence number of the downward re-routing channel from z, whether it runs back to the U-turn switch or towards the destination, must therefore be larger than the sequence number of any upward re-routing channel that may have led a packet to z.

No packet forwarded in the network will ever reach a channel that has a sequence number that is lower than or equal to that of any channels it has previously traversed, and $R^{link}_{deterministic}$ is deadlock-free.

■

It is worth noting that resuming the use of a previously failed link once it is restored to service is also deadlock-free, because the numbering scheme that we have utilised here remains intact.

VII. Adaptive Routing

We have given deterministic routing functions capable of tolerating $k-1$ arbitrary faults while remaining both livelock and deadlock free. The nature of the deterministic routing

algorithm required us to use one/two extra virtual channel(s) on all links to guarantee deadlock freedom. When we now consider adaptive routing, the adaptivity gives us an extra degree of freedom, removing the necessity of using virtual channels to maintain deadlock freedom.

Whereas deterministic routing required a strict ordering of the paths to be tested from the U-turn switches, we can use the greater degree of freedom afforded by adaptive routing to simply ensure that we test all possible paths once, but in any order, allowing maximum adaptivity. To facilitate this, a re-routing vector of length k bits is required in the switches, with one bit for each of the upward links to be tested from the U-turn switch. By setting the bit in the vector that corresponds to the link on which a re-routed packet arrives, the U-turn switch can keep track of which upward links lead to faulty paths and which are not connected to any faults. A timeout mechanism is required to reset the rerouting vector after a period of U-turn inactivity in order to be able to tolerate transient faults. Note that the use of the re-route vector is only required to guarantee the tolerance of more than one fault. If only one-fault tolerance is required, it is sufficient to choose an arbitrary upward link from the U-turn switch.

A. Link Faults

Using the foregoing reasoning as a basis, the adaptive dynamic fault-tolerant algorithm, $R^{link}_{adaptive}$, for tolerating up to and including $k-1$ link faults is presented below. The steps of this algorithm are marked in Figure 2 by numbers corresponding to the steps in the algorithm.

$R^{link}_{adaptive}$

1) ...
2) ...
   a) ...
   b) ...
3) The U-turn switch sets the bit in its re-routing bit vector that corresponds to the link on which the packet arrived (it is not necessary to try to use this link for forwarding the re-routed packet, because we know the path is broken).
4) The U-turn switch adaptively chooses one of its upward links that does not have its corresponding bit set in the bit vector and subsequently forwards the packet up this link. If there are fewer than k link faults, fewer than k of the bits will be set. Therefore, the packet will be forwarded as long as there are fewer than k link faults. The algorithm is then repeated from step 2.
5) A U-turn switch that receives a packet from an upward link in which all other bits in the re-routing vector are set discards the packet.

In the final stage of the algorithm, the U-turn switch may choose to inform the source node of its inability to find a correct path in order to prevent it from transmitting more packets.

One important feature of the above algorithm is that relative to the original adaptive routing for the fault-free case, only the re-routing in step 2 and the selection function (steps 1, 3 and 4) need to be modified. The only additional functionality that is needed for temporary faults is a timer that zeroes out the re-route vector after a given time interval. Note that this was not required for the deterministic routing algorithm, since it did not store any state information in the switches.

Figure 6 shows (i) a more detailed view of re-routing in a switch group that has multiple faults and (ii) the state of the re-routing vector. The connectivity, and livelock and deadlock freedom for the algorithm with $k-1$ link faults is shown in [17] (or Appendix A).

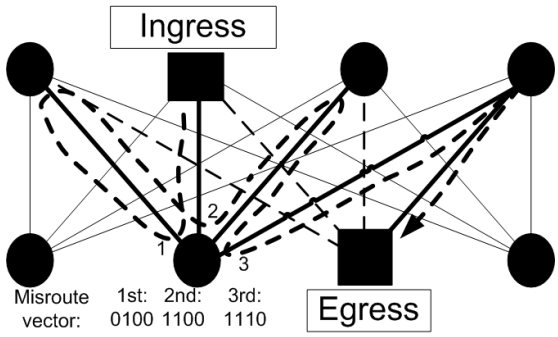


Figure 6. A combination of $k-1$ faults with re-routing in a group in a $k=4$ network.The terms 'ingress' and 'egress' refer to the switches where the packet enters and leaves the switch group, respectively. The numbered U-turns refer to the corresponding states of the re-route vector (listed next to the U-turn switch) as the U-turns are performed.

B. Switch Faults

Switch fault tolerance with adaptive routing is more complex than link fault tolerance with adaptive routing because of the number of possible paths that can be taken by a packet. Hence, in addition to re-routing down two tiers which following from Lemma 4 guarantees connectivity with less than k switch faults, we require the port field as for deterministic routing.

Below we list the routing algorithm $R^{switch}_{adaptive}$ to tolerate $k-1$ arbitrary switch faults. $R^{switch}_{adaptive}$ is illustrated in Figure 7.

The fault tolerance for $k-1$ switch faults is guaranteed by the fact that every possible U-turn switch u at the lower tier in the two-hop switch group has at least one upward link that leads to a switch in the set F, which again is connected to the switch T, as defined in the proof of lemma 3. T is not connected to any faulty switch through its downward links, and it will therefore never re-route any packet down to u. Consequently, there will always be one upward port in any U-turn switch for which the corresponding bit in the re-route vector will always be zero.

$R^{switch}_{adaptive}$

1) In the upward phase (towards tier 0), all upward links are supplied by the routing function and one is selected as the packet's outgoing link. If any of the links are faulty, the selection function simply does not choose it. This will result in the packet being forwarded as long as there are fewer than k link faults. When the packet has reached a switch with the packet destination in its subtree, the upward phase ends and is followed by the downward phase.

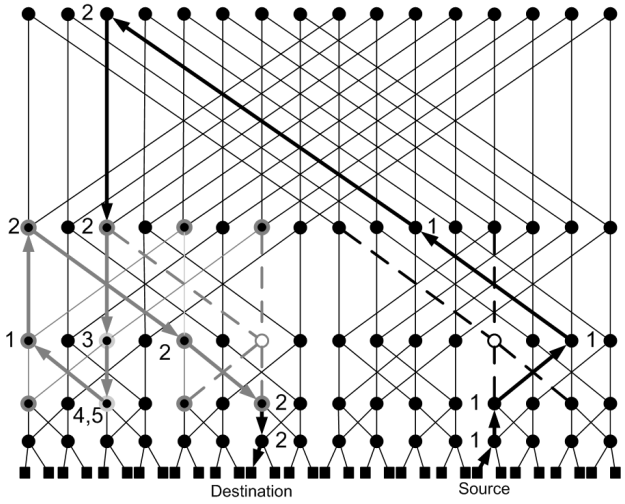


Figure 7. Re-routing in a 2-ary 5-tree
The bold, whole links show the path from the source to the destination, re-routing around a fault both in the upward and downward direction. The switches that are denoted as open circles with their corresponding dashed links have failed, and the switches and links with the grey outlines belong to a two-hop switch group. The numbers correspond to the points in the routing algorithm.

2) In the downward phase (towards tier $n-1$), only one link is supplied by the routing algorithm; namely, the link on the shortest path from the current switch to the destination. If this link is faulty, the following actions are performed:
   a) If the packet came from the switch tier above (from $l-1$ to l), another arbitrary downward link is selected, which forces the packet to be re-routed. This will result in the packet being forwarded as long as there are fewer than k link faults.
   b) If the packet came from the switch tier below (from $l+1$ to l), the packet is re-routed back down to the same lower-tier ($l+1$) switch again. This will make a re-routed packet in a switch group with more than one fault follow the path illustrated in Figure 6. This is necessary to avoid livelock.
3) A switch that receives a packet from a port connected to a switch at the tier below and which has a shortest path fault-free downward link, ensures that the *port*-field is -1 and forwards the packet downwards.
4) A switch that receives a packet from a port connected to a switch at the tier above, for which it has no downward path takes the following actions:
   a) If the *port*-field equals -1 it is set to -2 and another arbitrary downward link is selected, which forces the packet to be re-routed further down. This will result in the packet being forwarded as long as there are fewer than k link faults.
   b) If the *port*-field equals -2 this switch is a U-turn switch. The switch sets the bit in its re-routing bit vector that corresponds to the link on which the packet arrived (it is not necessary to try to use this link up for forwarding the re-routed packet, because we know the path is broken).
   c) If the switch is a bottom-tier switch (at tier $n-1$) it becomes a U-turn switch and sets the bit in its re-routing bit vector that corresponds to the link on which the packet arrived to guarantee link-fault tolerance. The *port*-field is set to -1, and the switch adaptively chooses one of its upward links to forward the packet.
   d) If the *port*-field does not equal either -1 or -2, forward the packet down through the port indicated by the *port*-field, which is reset to -2.
5) The U-turn switch adaptively chooses one of its upward links that does not have its corresponding bit set in the bit vector and subsequently forwards the packet up this link. If there are fewer than k link faults, fewer than k of the bits will be set. Therefore, the packet will be forwarded as long as there are fewer than k link faults.
6) A switch that receives a packet from a port connected to the lower tier in which the *port*-field equals -2 stores the incoming port number in the *port*-field and adaptively chooses one of its upward links. The algorithm is then repeated from step 2.
7) A U-turn switch that receives a packet from an upward link in which all other bits in the re-routing vector are set and the *port*-field is -2 discards the packet. All upward links of the U-turn switch have been tested and there is no available path to the destination from this switch. This may only happen when there are k or more faults.

The connectivity and deadlock freedom properties of this algorithm for $k-1$ switch faults is presented in [17] (or Appendix A).

VIII. Endpoint Dynamic Rerouting

Because of the lack of any other dynamic local rerouting algorithms to compare with in the evaluation presented in the next section, we compare our dynamic local rerouting algorithm for deterministic dynamic local re-routing (DDLR) and adaptive dynamic local re-routing (ADLR) with a fault-tolerance algorithm that we call deterministic dynamic reconfiguration (DDRC). DDRC is one of the more straightforward ways of tolerating faults dynamically in a deterministically routed fat-tree. It may be used for both source routing and distributed routing. For source routing, every network element that is connected to the element that fails broadcasts information about the failure to all processors in the network. Each processor may then remove all paths that involve the faulty element from their routing table, or remove the faulty element from its network graph and compute new routing tables using standard fat-tree routing, thus avoiding the fault. Distributed routing tables in interconnection networks are usually set up by a management unit. In this case, the nodes that are connected to the failure send information to the management unit, which calculates new tables without the faulty elements and uploads these to the switches.

A. Hybrid Dynamic Rerouting

Since dynamic local re-routing (DLR) and DDRC have two distinctly different methods of operation, it is possible to combine the two into a hybrid approach. When both methods are used simultaneously, DLR may re-route packets while the processing nodes or a management unit is being informed of the fault event. This will reduce the number of packets lost per fault. However, the probability that the routing algorithm will keep the network connected is dictated by DDRC. The reason is that all paths that contain faults will be removed by the algorithm. Therefore, DLR will only operate in the interval between (i) the faults being discovered and (ii) the routing table of the switches being updated by the management unit and all packets that followed the old routes being drained from the network. For this hybrid approach, all performance metrics that we evaluate in the next section are the same as for DDRC, except for packets sunk per link fault, which will be the same as for DLR. Therefore, it will not be evaluated through any specific simulations.

IX. Evaluation

In this section, we evaluate the performance of the proposed *dynamic local re-routing* method (DLR) with both deterministic routing (DDLR) based on $R^{link}_{deterministic}$ and adaptive routing (ADLR) based on $R^{link}_{adaptive}$. The most prominent feature of the DLR mechanisms is the speed of their response to failures, which results in very few packets being lost. However, the price of this almost instantaneous fault tolerance is paths in the network that are less optimal, and thus reduced performance. We compare the DLR mechanisms against the *deterministic dynamic reconfiguraton* method (DDRC) that we presented in Section VIII. We will use an implementation of DDRC close to that which can be achieved in Infiniband. Upon discovering a faulty port, the switch that makes the discovery sends a message to the Infiniband subnet manager to inform it of the failure. The subnet manager then generates new routing tables and distributes these to all switches in the network.

We evaluate the effects that the two paradigms have on performance, to illustrate the difference between using optimal paths (DDRC) and suboptimal paths combined with deterministic (DDLR) or adaptive routing (ADLR). We compare the three methods with respect to performance degradation with link faults; hence, DDRC is configured with only one virtual channel, even though DDLR in fact requires two. In this manner we may use the performance degradation as a measure of the cost of the fast failover of ADLR and DDLR compared to DDRC. We also compare the probabilities that the three algorithms will keep the network connected beyond their guaranteed connectivity limit of $k-1$ faults. For this comparison the probability will be the same for DDLR and ADLR as they both rely on the same re-routing mechanism to maintain connectivity.

We have only included the evaluation for link faults in this paper. Switch faults will have a more severe performance degradation than link faults because more paths are taken out of commission by a single fault, and the re-route path is longer than for link faults. However, the general conclusions we draw at the end of the evaluation are valid for both types of faults.

A. Simulation Environment

We have conducted a series of network simulations in which the experiments were run with an increasing number of link faults, (from 0-10) in order to examine how the methods perform as the network degrades. The simulations were performed in an event driven simulator developed in-house at Simula Research Laboratory. It is based upon the J-sim framework [25]. In our evaluation, the link bandwidth was 2.5 Gb/s. However, the links used the 8b/10b encoding scheme; hence, the maximum effective bandwidth for data traffic was limited to 2 Gb/s. The size of a virtual output queue for a VC is 512 bytes, which is sufficient to allow buffering of up to two 256 byte packets, which in turn is close to the packet size/buffer space ratio that is often found in real hardware. The send queue of the processing nodes was set at 1.3 MB, which is sufficiently large to inject the required traffic into a variably loaded network.

Two topologies were simulated: the 4-ary 3-tree and the 2-ary 6-tree. These topologies consist of 48 (64) and 192 (64) switches (computing nodes), respectively. They have a moderate number of network elements, which allows the required quantity of simulations to be run within a reasonable time. Furthermore, the effects of faults in these small-scale networks represent upper bounds on the impact on performance in larger networks, because the impact of any single fault in a large network will be smaller than in a small network. Regarding the traffic pattern, both uniform and hotspot traffic was validated, but only the results were uniform traffic have been included as this clearly illustrates the performance of the algorithms. Packet generation was governed by a normal approximation of the Poisson distribution. Further, the packet size was 256 bytes. The link-transfer rate was 128 bytes per cycle. With an effective link bandwidth of 2 Gb/s, this gives 1 second = 1953125 simulation cycles.

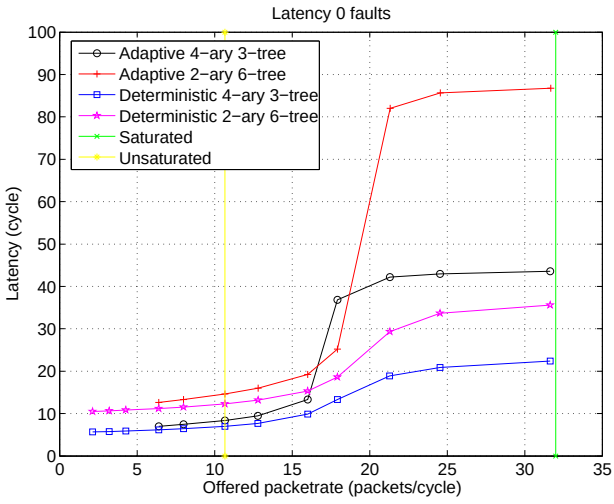


Figure 8. Saturated and unsaturated loads with uniform traffic.

Two load cases were evaluated for increasing numbers of link faults. The exact loads chosen for the evaluations are marked by the vertical lines in Figure 8 for uniform traffic: i) saturated, which was slightly above the saturation

point (relative to the fault-free network), and ii) unsaturated, where the load was 30% below the saturation point. The figure shows the throughput and latency for an increasing network load for the two traffic distributions, with adaptive and deterministic routing for both tested topologies. The figure clearly shows when the networks saturate, and the vertical lines mark the selected unsaturated and saturated loads we use in the following experiments.

Each data point in the performance figures is the average of 500 independent simulation experiments. For each experiment, we let the simulation run until the average latency had stabilized, before the measurements were started. Thereafter, the simulation was run for another 10000 simulation cycles.

B. Evaluation Results

We start the evaluation by considering the degree of network utilisation the various routing algorithms are able to maintain as more and more link faults are introduced into the networks. The throughput obtained by the three methods, DDLR, DDRC, and ADLR, in a 4-ary 3-tree and a 2-ary 6-tree fat-tree topology for saturated and unsaturated loads is presented in Figure 9. The x-axis in these figures represents the number of link faults introduced into the network, and the y-axis is the average number of packets per cycle accepted by the network injected by the end nodes. The vertical line at the points 3 and 1 on the x-axis depict the $k-1$ fault tolerance limits in all three evaluated algorithms.

If we consider the case for the 4-ary 3-tree with uniform traffic in Figure 9(a), we notice that for zero faults ADLR achieves a slightly lower saturated throughput than the DDLR and DDRC methods. Recall that without faults DDLR and DDRC are in essence the same routing algorithm. This is because for a uniform traffic pattern, a well-balanced deterministic routing algorithm will outperform adaptive routing [7]. However, as the number of link faults increases, we see that the saturated throughput for DDLR quite rapidly decreases; already for one link fault, ADLR displays higher throughput than DDLR. Furthermore, we see that the saturated throughput of DDRC is higher than for the two other algorithms for any number of faults (except for zero faults where it is identical to DDLR). Of the DLR methods, ADLR is in other words better at maintaining high throughput with link faults, showing only a slightly higher decrease in throughput as the number of faults increases than DDRC. The reason for this is that the path used for re-routing around the link faults quickly becomes a bottleneck in the network. This severely impacts the performance of DDLR, while ADLR benefits from its adaptivity in being able to redirect traffic to other healthy paths. This view is further strengthened if we consider the unsaturated loads for the three methods. In this case all algorithms are able to maintain the same throughput as without link faults, except for DDLR which also here shows a throughput reduction as we pass three link faults.

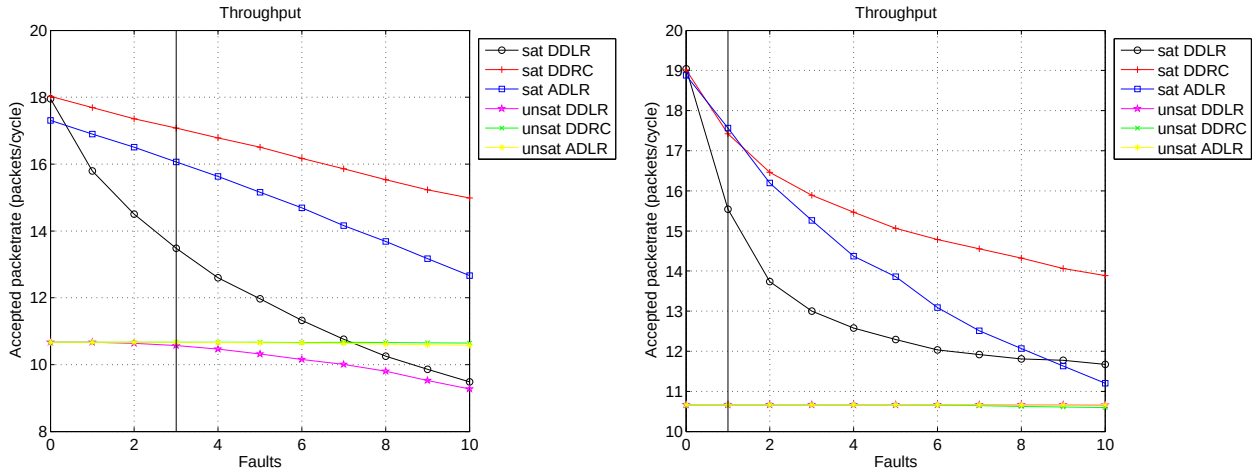
Moving on to the 2-ary 6-tree in Figure 9(b) the situation is somewhat different. First, note that since $k=2$ the algorithms can only guarantee toleration of one link fault. We see that for saturated throughput without link faults all three algorithms all attain almost equal performance. The 2-ary 6-tree consists of switches with a significantly lower number of ports than the 4-ary 3-tree, limiting the number of available paths for utilisation by ADLR. The difference between deterministic and adaptive routing will therefore be less noticeable, although the adaptive throughput is slightly lower. However, as the number of faults increases, we again see that the throughput of DDLR is more dramatically affected by link faults than both ADLR and DDRC. Another interesting point is that as we pass nine link faults the throughput of ADLR slips below the throughput of DDLR. With this large number of link faults there is a high probability of encountering a number of link faults on either of the available paths in the network. As a consequence of this, the relative performance difference between the various paths becomes smaller as in the case without link faults, and the performance of ADLR will be lower than DDLR. The method with the best performance is still DDRC which simply removes the faulty paths from the routes without having to re-route traffic. However, as we will see later in the evaluation, the involvement of a central entity, the subnet manager, causes a severe time penalty to this operation. This will greatly reduce the effectiveness of DDRC, especially for short-lived faults.

It is evident from Figure 8 that the 2-ary 6-tree has a greater forwarding capacity with uniform traffic than the 4-ary 3-tree for the same number of endpoints. For unsaturated throughput the number of link faults we have introduced is not sufficient to cause network congestion, and thus throughput reduction. However, careful examination will reveal the same behaviour as for the 4-ary 3-tree, for 10 link faults the throughput of DDLR shows a slight decrease.

Let us next consider the number of packets discarded by the network from the time that the fault occurs until it is tolerated, which is the strongest point in favour of the DLR methods. For DDLR and ADLR, this is plotted in Figure 10. The number of packets lost per link fault is plotted along the y-axis, while the increasing number of link faults is plotted along the x-axis. Note that interconnection networks do not discard packets, so the packet loss accounted here is exclusively caused by the link failures.

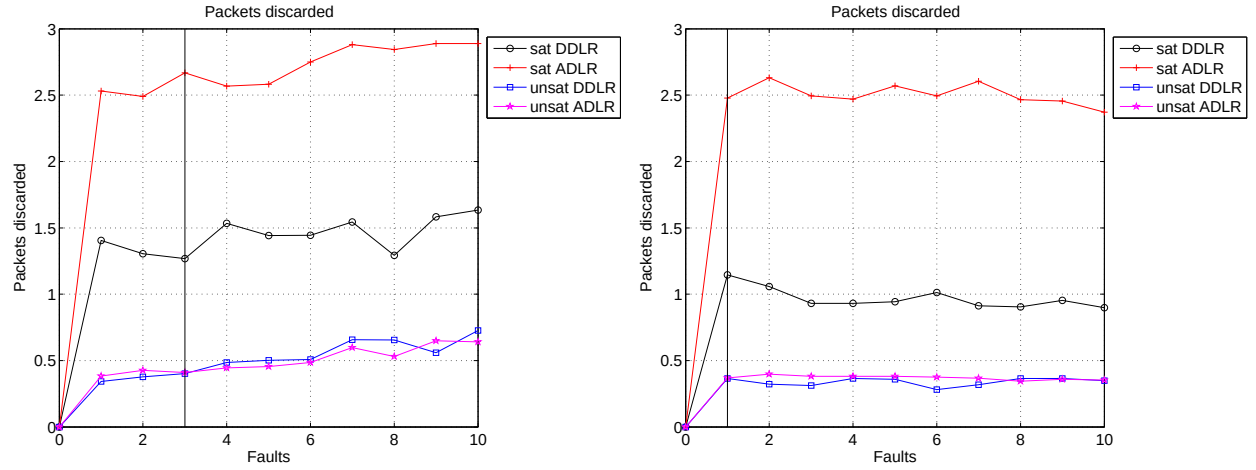
We see that for any number of faults, the number of packets sunk because of each link fault for the saturated case is about 1.5 and $2.5-3$ in the 4-ary 3-tree with uniform traffic (Figure 10(a)) for DDLR and ADLR respectively. For the unsaturated case it increases from 0.4 to 0.6 for one through ten faults for both DDLR and ADLR. Recall that the DLR methods only loose packets that are either currently in transit over the failing link or queued for the failing link.

We found that the simulation time we used was insufficient to encompass the reconfiguration of the DDRC method. Hence, we were forced to resort to calculating analytically the number of packets lost during the reconfiguration. To achieve this, we first approximate the time it takes to discover a fault, compute new forwarding tables, and distribute them. Using as a basis the work done by Bermudez et al. [2], which evaluates the time it takes for the Infiniband subnet manager to perform these tasks, we find that the time taken to distribute the new forwarding tables is about 0.03 seconds in a 32-switch network. The full time taken to reconfigure the network is



(a) 4-ary 3-tree, uniform traffic (b) 2-ary 6-tree, uniform traffic

Figure 9. Throughput for a 4-ary 3-tree and 2-ary 6-tree. Both uniform and hot-spot traffic is depicted, for the three methods: DDLR, DDRC, and ADLR. The faults range from 0 to 10.



(a) 4-ary 3-tree, uniform traffic (b) 2-ary 6-tree, uniform traffic

Figure 10. Packets discarded for a 4-ary 3-tree and 2-ary 6-tree. Both uniform and hot-spot traffic is depicted, for two of the three methods: DDLR, and ADLR. The faults range from 0 to 10.

about 0.5 seconds, most of which is used for routing table calculation. In the calculations we assume that this calculation is instantaneous, because the fat-tree routing algorithm is easy to set up. In addition, the subnet manager expends no time on discovering the topology change, because it is informed of the fault by the switches. For the sake of simplicity we assume the information is transmitted instantaneously. Therefore, 0.03 seconds is a lower bound on the time it will take to reconfigure a 32-switch network.

First, the 0.03 seconds required for updating all forwarding tables from the subnet manager corresponds to approximately 58593 simulation cycles. Second, to calculate the number of packets lost in the presence of one link fault, it is necessary to determine the packet rate over a single link in the fat-tree. This depends on (a) whether the link is at the bottom or top link tiers of the fat-tree, and (b) the total accepted packet rate of the network, which can be seen from Figure 9 for the saturated case with zero faults at approximately 18 packets per cycle. Based on this we can calculate that the bottom tier link has a packet rate of 0.56 packets per cycle, and an upper tier link carries 0.42 packets per cycle.

Multiplying this by the number of cycles in 0.03 seconds yields a total of 32812 and 24704 packets lost per link fault located at the lower or upper link tiers respectively; about 20000 times as many packets as DDLR and 10000 times as many as ADLR. Keep in mind that the reconfiguration time of 0.03 seconds was based on a 32-switch network, which is smaller than the one considered here, and it was assumed that it would take zero time to inform the subnet manager of the failure and to calculate new tables. Hence, the actual packet loss for DDRC may be significantly larger than that calculated here, given the packet size. In that respect, the calculation is optimistic compared to a real case.

For the 2-ary 6-tree the situation is much the same, with ADLR losing more than twice as many packets per link fault as DDLR with saturated traffic. For unsaturated traffic the packet discard count is quite low since an unsaturated network has minimal queuing time, and thus fewer packets are likely to be

stuck in disconnected queues.

The reason for the increased packet loss of ADLR compared to DDLR is that ADLR through its adaptivity distributes traffic more evenly in the network, thus forcing more buffers to be occupied. Hence, there is a higher probability of there being a large number of packets in the buffers that are connected to the link that fails. The average packet loss will therefore be somewhat higher.

The fundamental property of any fault tolerant routing algorithm is the ability of the algorithm to keep the endpoints connected when there are faults present in the network. In Figure 11 we have plotted the probability (y-axis) of the routing algorithm keeping the network connected after the occurrence of a given number of faults (x-axis) for the 4-ary 3-tree (Figure 11(a)) and the 2-ary 6-tree (Figure 11(b)) for the three algorithms. Note that the probability of maintaining connectivity is the same for the two DLR methods, ADLR and DDLR, as they both use the same re-routing mechanism to forward the packets, and that this probability is different from the probability of the network being physically connected. The difference that can be observed between the two methods in the figures is therefore a result of statistical variation for the 500 samples we have simulated. Recall that the 4-ary 3-tree is able to guarantee a connected network when there is less than four link faults. The figure shows that for the 4-ary 3-tree, all algorithms managed to maintain connectivity for all 500 tested combinations of 4 link faults. Although such connectivity cannot be guaranteed, the probability of four link faults disconnecting a 4-ary 3-tree is very low. As the number of link faults increases towards 10, the probability for all three algorithms decreases by roughly the same amount down towards 97%. For the 2-ary 6-tree, on the other hand, the algorithms can only guarantee the toleration of one link fault. We see from the figure that already at two link faults none of the algorithms maintain a 100% probability of connectivity. Furthermore, we see that the probability of connectivity for DDRC stands out by decreasing more rapidly than for the two DLR algorithms. Local re-routing is in other words capable of tolerating a larger number of fault combinations for a given number of faults than reconfiguring the network to avoid the faulty paths in fat-trees.

Finally, we will consider the probability of the networks becoming deadlocked when operating outside the limit specified by the theory in the previous sections. For ADLR this is any number of faults eyond $k-1$. Recall that the deadlock freedom theory for $R^{link}_{adaptive}$ only has validity when there are less than k faults. $R^{link}_{deterministic}$, on the other hand, is guaranteed to be deadlock free for any number of faults as long as one/two additional virtual channels are used when re-routing. However, without these virtual channels, deadlock freedom can only be guaranteed with one fault. Any number of faults larger than this carries the risk of deadlocking the network.

To generate these figures we have run 500 simulations for each of 4,7, and 10 link faults for the 4-ary 3-tree. We have let each simulation run for 200 000 simulation cycles and recorded the time at which the network became deadlocked, if at all. The resulting plots will asymtothically approach the probability of deadlock for the given number of faults as the time approaches infinity. We have obviously not be able to let time approach infinity, but it is still possible to get a general idea of the probability from the figures presented. For the 4-ary 3-tree (Figure IX-B) we see that for four link faults, the lowest number of link faults for which we do not guarantee deadlock freedom, the probability of deadlock slowly approaches 1%. For seven link faults the probability approaches 7%, and for 10 faults the probability of deadlock has climbed to about 20%. Comparing this to the probability of connectivity discussed previously, we see that even though the 4-ary 3-tree remained connected for all combinations of four link faults we tested, there is a 1% probability of the network becoming deadlocked. Similarly, for 10 link faults the probability of maintaining connectivity was about 97%, but the probability of deadlock for this case is 20%.

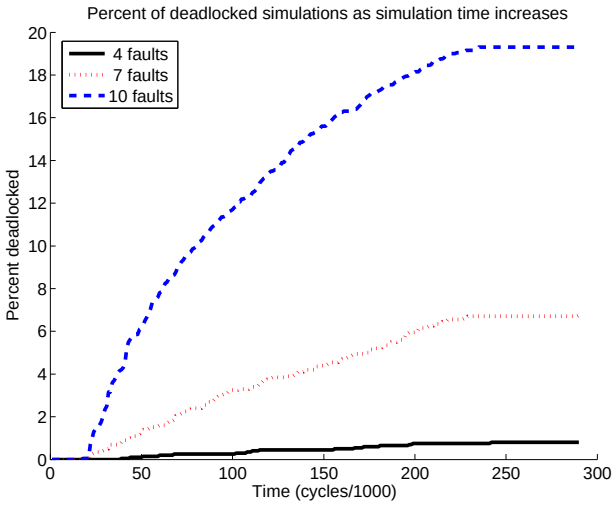


Figure 12. Probability of deadlock beyond $k-1$ faults with adaptive routing. For a 4-ary 3-tree was four, seven, and 10 faults.

X. Conclusion

The size and complexity of modern high-performance computer systems exacerbates the need for efficient fault-tolerant mechanisms to guarantee high system performance. In this paper we have presented a fault-tolerance mechanism based on dynamic local rerouting, aimed at the much utilised fat-tree topology. The fault tolerance algorithm can transparently tolerate up to and including $k-1$ arbitrary link faults, where k is number of ports in one direction of a switch in the fat-tree. Depending on whether adaptive or deterministic routing is utilised, deadlock freedom can be guaranteed without using additional resources, or with one extra virtual channel, respectively.

Simulation experiments show a graceful degradation in performance as the number of link faults in the system increases. For larger fat-trees with higher radix switches the effect of any link fault in using dynamic local rerouting is believed to be significantly smaller. For the relatively small topologies we tested, the degradation is higher than for a simple reconfiguration mechanism to which we compare the results. However, the speed and transparency of the mechanism has a significantly

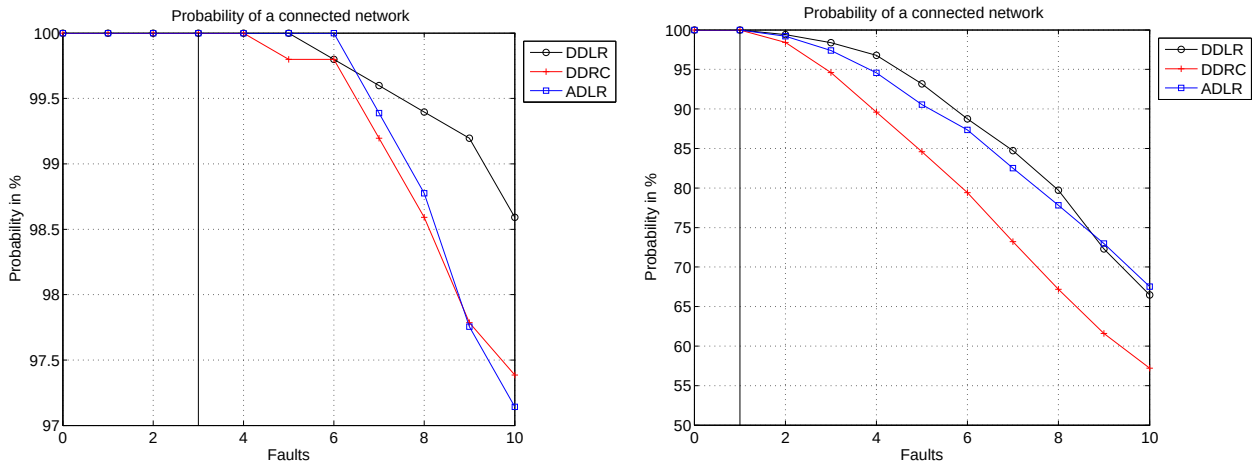


Figure 11. Probability of connectivity for a 4-ary 3-tree and 2-ary 6-tree. For the three methods: DDLR, DDRC, and ADLR. The faults range from 0 to 10.

lower impact on network traffic. A high-speed reconfiguration mechanism may lose up to 20,000 times more packets than our dynamic local rerouting mechanism. Finally, the probability of maintaining network connectivity for all end nodes is slightly higher using dynamic local rerouting compared to reconfiguring the fat-tree. Therefore, a careful combination of dynamic local rerouting with reconfiguration as in the hybrid approach will yield a fault tolerance mechanism with high-performance, virtually no packet loss, and high probability of connectivity beyond the guaranteed connectivity region.

APPENDIX A

A. Deadlock and Livelock Freedom with One Fault

We have shown that every switch has a connected path to every destination when re-routing down one switch tier with $k-1$ arbitrary link faults. How we may configure the routing algorithms to find these paths while guaranteeing livelock and deadlock freedom is the topic of the next sections. There are separate sections for deterministic distributed and source routing, and adaptive routing, since there are significant differences between how the routing algorithms must be set up for the different cases. Before we proceed with this, however, we will show that adding the re-routing mechanism in the downward phase is deadlock and livelock free if there is just one link fault in the network, regardless of whether we have adaptive or deterministic routing.

It is well-known that a network may deadlock if there exists a chain of channel dependencies that form a cycle [5], so that it is possible to follow the dependency chain from one specific channel and subsequently arrive at the same channel (although this is not necessarily the case for adaptive routing). If there is no such cyclic dependency chain, there is no deadlock.

Theorem 2: Re-routing around a single link fault will never deadlock or livelock when there are no packets in the network that have been re-routed around any prior fault.

Proof: Suppose the link between upper-tier switch S_u at tier l and lower-tier switch S_v at tier $l+1$ in group G_1 has failed. In the upward phase, simply selecting a different upward link to avoid the fault will not cause additional dependencies to those already present from the up/down routing. For the downward phase, at least one of the other lower-tier switches (except S_v) of G_1 may be a U-turn switch, S_uturn. Next, suppose the network is deadlocked, so there is a cycle in the channel dependency graph of the network. This cycle must involve a dependency from the channel from S_u to the u-turn switch S_uturn to the channel from S_uturn to an upper-tier switch of G_1, S_up, and a chain of dependencies back to S_u. There can be no dependency cycles within G_1 because no packets are re-routed to S_u so $S_u \neq S_up$. Outside G_1, the only possible dependency chain from S_up to S_u is upwards through a switch S_r higher up in the tree, and then downwards. This downward portion of the dependency chain will never go through G_1; otherwise, S_r would be able to reach the same destination through two downward links, which is impossible. Hence, this dependency chain must involve another U-turn for it to be able to return back to S_u and S_uturn. However, given that there is only one fault in the network and no packets following paths created by previous faults, there are no U-turns outside G_1 and thus there can be no cycle.

Livelock freedom is guaranteed since from any switch connected to a fault in the downward direction all other downward links leads to a U-turn switch. Similarly, all upward links from a U-turn switch (except the link from which the re-routed packets arrive) have connected up/down paths to the destination. Any re-routed packet will therefore never encounter the same fault twice, and is always forwarded on the shortest connected path to the destination. ■

B. An algorithm for calculating forwarding tables for $R^{link}_{deterministic}$

Algorithm 1 presents the pseudocode of the function to fill the routing tables of a switch for a destination in the fat tree. It requires that the routing function is able to take both input port and input VC as parameters in addition to the destination address, and returning output port and output VC. Recall that any VC belongs to either NL or ML. The only additional

mechanism required in the switches is the ability to select (i) an arbitrary output port in the correct direction when a faulty link is encountered and the packet is in NL, and (ii) the incoming ports as the output port when a faulty link is encountered and the packet is in ML.

Algorithm 1 $R^{link}_{deterministic} - table$

```
1: global array sequence[k]
2: global array route[][][]
3: sequence[incomming port p]=next upward port to test
4: sequence[0]=first port in the sequence
5: VC1 belongs to normal VL
6: VC2 belongs to re-routing VL
7: route[destination][incomming port][incomming VC]=outgoing port, outgoing VC
8: function ROUTEDESTINATION(Switch s,Destination d)
9:     if d not reachable downwards then
10:        select upward port dp ▷ should be balanced in some way
11:        for all downward ports p in s do
12:            s-> route[d][p][VC1] = dp, VC1 ▷ set up regular routes for all downward ports
13:        end for
14:        for all upward ports p in s do ▷ u-turns ▷ for the first u-turn
15:            s-> route[d][p][VC1] = sequence[0], VC2 ▷ for subsequent u-turns through the same switch
16:            s-> route[d][p][VC2] = sequence[p], VC2
17:        end for
18:    else
19:        dp=downward port to d
20:        for all ports p in s except dp do ▷ set up regular routes for all incomming ports
21:            s-> route[d][p][VC1] = dp, VC1
22:            if p is downward port then ▷ Route downwards after the u-turn
23:                s-> route[d][p][VC2] = dp, VC2
24:            end if
25:            if p is upward port then ▷ Continue downwards below the tier of the fault and return to normal layer
26:                s-> route[d][p][VC2] = dp, VC1
27:            end if
28:        end for
29:    end if
30: end function
```

C. Livelock freedom and connectivity for $R^{link}_{deterministic}$

We now prove livelock freedom and the $k-1$ fault tolerance of $R^{link}_{deterministic}$.

Lemma 6: $R^{link}_{deterministic}$ is connected and livelock free with $k-1$ arbitrary link faults.

Proof: Lemma 2 guarantees a path from every switch to every destination using the re-routing function implemented in $R^{link}_{deterministic}$. It is therefore sufficient to show that the routing algorithm is livelock free. The number of links is finite; hence, a livelock requires that a packet is forwarded in a loop. There must therefore exist a set of switches that the packet may traverse 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 re-routing around a fault. However, the ordered sequence D utilised in point 4 of $R^{link}_{deterministic}$ and the re-routing in point 2b of $R^{link}_{deterministic}$ guarantees that all the upper-tier switches in the switch group containing the faults are used as next hops at most once; hence, none of these can be involved in a livelock. Given that there are fewer than k link faults, it is guaranteed that there is a path that moves the packet out of the switch group and one tier lower down towards the destination. Consequently, every time a packet is re-routed, it will find a path that brings it one tier closer to its destination, and the algorithm is livelock free. ■

D. Livelock freedom and connectivity for $R^{switch}_{deterministic}$

Lemma 7: $R^{switch}_{deterministic}$ is connected and livelock free with $k-1$ arbitrary link or switch faults.

Proof: Lemma 4 guarantees a path from every switch to every destination. It is therefore sufficient to show that the routing algorithm is livelock free. The number of links is finite; hence, a livelock requires that a packet is forwarded in a loop. There must therefore exist a set of switches that the packet may traverse 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 re-routing around a fault. However, the ordered sequence D utilised in point 4 of the algorithm, the port field that guarantees in points 4c and 5 that a packet always returns to the same u-turn switch, and the re-routing in point 2b, guarantees that all the upper-tier switches in the switch group containing the faults are used as next hops at most once; hence, none of these can be involved in a livelock. Given that there are fewer than k switch faults, it is guaranteed that there is a path that moves the packet out of the switch group and one tier lower down towards the destination. Consequently, every time a packet is re-routed, it will find a path that brings it one stage closer to its destination, and the algorithm is livelock free. ■

E. Deadlock Freedom for $R^{switch}_{deterministic}$

To achieve deadlock freedom with deterministic routing and the fault tolerance algorithm presented above, we require the use of three virtual layers: (i) a normal virtual layer where most of the packet forwarding takes place, (ii) a re-routing layer (ML1) that contains the packets currently being re-routed in the lower half of the two-hop switch group, and iii) a second re-routing layer (ML2) containing all packets being re-routed in the upper half of the two-hop switch group.

The transition from the normal layer to ML1 takes place in the U-turn switches. Each time a packet performs a U-turn it enters ML1. The transition from ML1 to ML2 takes place in the switch at the tier above the U-turn switch when re-routing upwards, and the transition from ML2 to ML1 takes place in a switch at the same tier when re-routing downwards. The transition from ML1 back to NL takes place in any switch

at the lower tier that is not the u-turn switch. This will only occur after the fault leading to the u-turn has been negotiated.

In order to show that $R^{switch}_{deterministic}$ is deadlock-free we will use the same technique as we used for $R^{link}_{deterministic}$. We introduce a numbering scheme which we apply to all channels in the fat tree. Because of the complexity of the algorithm, the numbering scheme will also be more complex.

The fat-tree configured for switch fault tolerance consists of six types of channels: upward and downward channels in NL, upward and downward channels in ML1, and upward and downward channels in ML2. Let N^u_s be the set of upward channels in NL at link tier s and N^d_s the set of downward channels in NL at link tier s. $M1^u_s$ and $M1^d_s$ are the set of upward and downward channels at tier s in the first re-routing layer, and $M2^u_s$ and $M2^d_s$ is the set of upward and downward channels at tier s in the second re-routing layer. u' is the sequence number assigned to the channel u. The assignment $u' = j$ where j is an integer greater than -1 and $u \in N \cup M_1 \cup M_2$ assigns the number j to the channel u. The link tier numbering is almost the same as we used for link faults. We number the channels as follows:

1) $u' = n - 2 - s, \forall u \in N^u_s, s \in 0 \ldots n-2$
   All upward channels in NL are assigned the reversed indexing of the link tiers.
2) $u' = v' + 1 \forall u \in N^d_0, \forall v \in N^u_0$.
   All downward channels in NL at the topmost link tier are assigned a number that is 1 greater than the upmost upward channels in NL.
3) $u' > v' \forall u \in N^d_s, \forall v \in M1^d_{s-1} \cup N^d_{s-1}, s > 0$.
   Every downward channel at tier s in NL is given a number that is larger than the downward channels in NL and all channels in the ML1 for all tiers higher than s.
4) $u' = v' + (i * 4) + 1, w' = v' + (i * 4) + 4 \forall u \in D \in M1^u_s, w \in D \in M1^d_s$, i is the index of u in the sequence D, $\forall v \in N^d_s$, u and w are part of the same link.
   Every ML1 channel is given a number larger than the normal downward channels at tier s, in an increasing order from $v' + 1$ corresponding to the index of the re-routing channel in the re-routing sequence D from the u-turn switch. For instance, assuming that the largest sequence number of all downward channels in NL at tier s is 10, the first upward M1 channel in the sequence to be tested from a U-turn switch at tier s is 11, the first downward M1 channel is 14, the second upward M1 channel is 15, the second downward M1 channel is 18, and so forth.
5) Let u, v, w be connected to the same switch at tier s. u and w are the upward and downward channels of the same link connected to an upward port, and v is an upward channel connected to a downward port. $u' = v' + 1, w' = v' + 2 \forall u \in M2^u_s, w \in M2^d_s, v \in M1^u_{s+1}$, u, v, w are connected to the same switch at tier s, $s >= 0$.
   Every upward channel in M2 connected to an upward port of a given switch at tier s is given an index that is one higher than the index of all the M1 channels connected to the downward ports of the same switch. We will show later that all the M1 channels will have the same index. Every downward channel in M2 for all links connected to the upward ports of the same switch is given an index one higher than the upward M2 channels on the same link.

Before we proceed with the formal proof of the validity of the numbering scheme and deadlock freedom, we must consider one final corollary, which shows that any upper-tier switch in a 2-hop switch group is connected through downward links only to switches that have the same index in the re-routing sequence D for any U-turn switch in that group. This means that all upward channels in M2 to the switch will have the same sequence number i, and all downward channels in M2 from the switch will have the same sequence number $i+1$ (This follows from rule 5).

Corollary 2: An upper-tier switch u in a 2-hop switch group G_2 is connected through downward links only to switches that have the same index i in the re-routing sequence D for all U-turn switches in G.

Proof: Consider a 2-hop switch group G_2 with its upper tier at tier l, and its lower tier at tier $l+2$. From Definition 1 , all switches at tier $l+1$ in G_2 are connected to port $w_{l+1} + k$ of the switches at tier $l+2$, where w_{l+1} is the $(l+1)$'th tuple of the switch number. For any given ordering sequence D, all switches at tier $l+1$ in G_2 with the same value in w_{l+1} will have the same index in the sequence D. Further, any switch at tier l in G_2 where $w_(l+1) = a$ for any a is only connected to switches at tier $l+1$ that also have $w_{l+1} = a$. Thus, any switch at tier l in G_2 is connected only to downward switches with the same index in D.

■

We now show that the proposed numbering scheme provides all channels in the network with a single number.

Lemma 8: Every channel in the fat-tree can be assigned a number that satisfies all five points of the numbering algorithm.

Proof: We will first consider the channels in NL. Every channel in the fat-tree belongs to a single link tier. Rule 1 guarantees that all upward channels in NL are assigned a number. The only requirement on these numbers is that they be the reverse order of the link tier indexing, which is trivial to guarantee.

For the downward channels in NL, rules 2 and 3 guarantee that every channel is assigned a number. From rule 4, the maximum number assigned to M1 channels at a tier l is limited by the number of channels in the re-routing sequence which is $4k$, so the maximum number assigned to any channel at a tier above tier l is $n - 2 + l + l * 4k$ (the numbers assigned to all upward channels in NL, plus the number assigned to all downward channels at each tier, plus all the numbers assigned to re-routing channels at each tier). Any downward channel in NL at tier l may therefore be assigned a number that satisfies rules 2 and 3, which is on greater than $n - 2 + l + l * 4k$.

Similarly, rule 4 guarantees that all M1 channels at all tiers are assigned a number. Rule 2 guarantees that all top-tier downward channels in NL are assigned the same number n. Assigning the number $n + i * 4 + 1$ to consecutive upward channels with index i in the re-routing sequence D, and

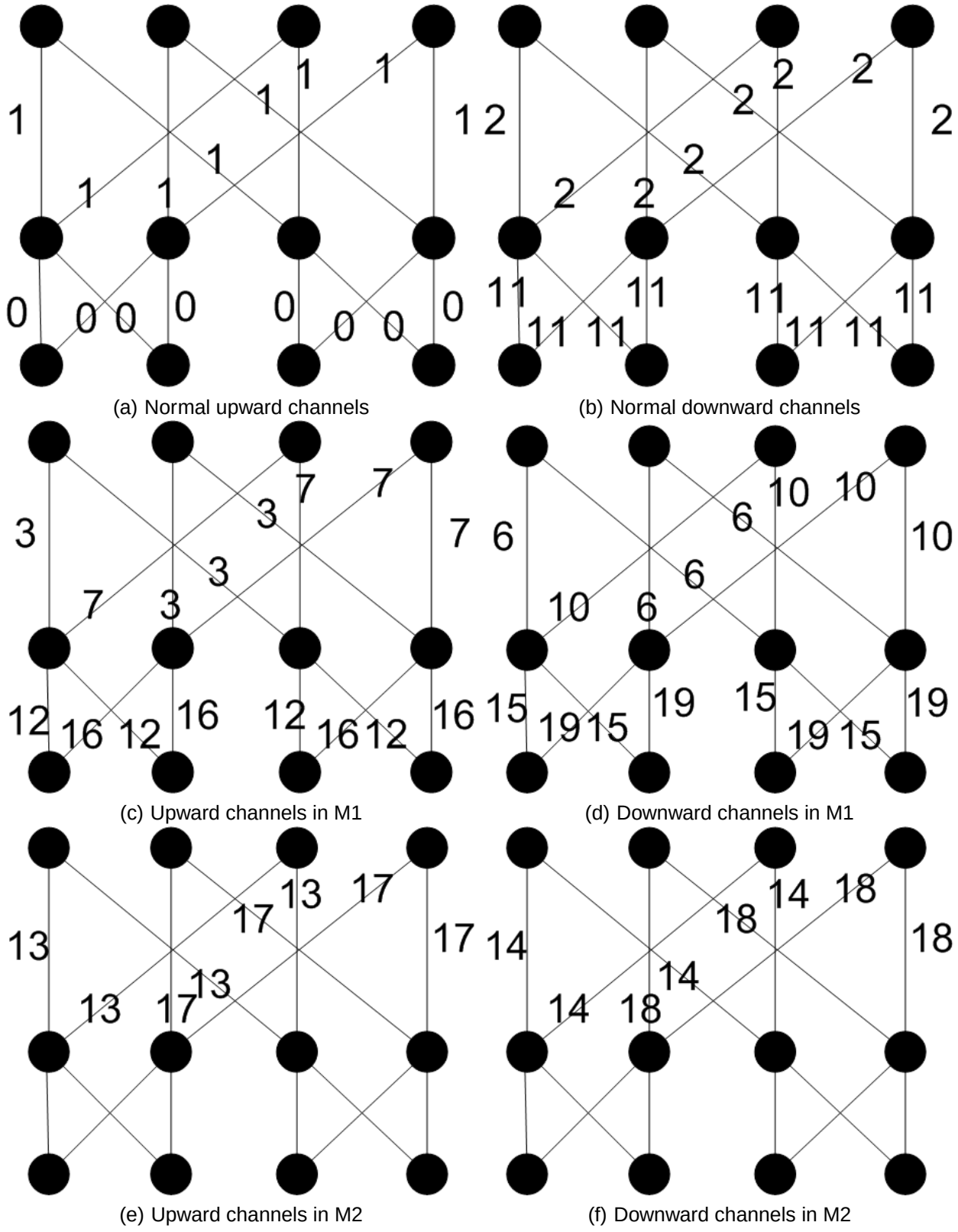


Figure 13. Channel numbering in a fat-tree for switch faults

Numbering of the different channels in the normal and re-routing layers, assuming that the ordered sequence D consists the upward ports from left to right.

assigning $n + i * 4 + 4$ to consecutive downward channels, combined with the fact that any M1 channel belongs to the re-routing sequence of one and only one U-turn switch, guarantees that we conform to rule 4 for the top-tier re-routing channels. For re-routing channels at subsequent lower link tiers, the same reasoning applies, except that the number of the downward channels in NL of each tier l is given by $n - 2 + l + l * 4k$, as shown previously.

Finally, rule 5 guarantees that all M2 channels at all tiers are assigned a number. It follows from Corollary 2 that all up/down channels in M2 that are connected to the same upper-tier tier switch in G_2 will be connected to middle-tier switches in G_2 that are connected only to downward links with the same index in D and therefore the same channel number. Assigning a sequence number to the upward M2 channels connected to the upward ports of a switch that is one larger than an arbitrary upward M1 channel connected to the downward ports of the same switch will therefore yield the samme number regardless of which of the upward channels in M1 are chosen. The same is the case for the downward M2 channels connected to the same switch. ■

This numbering scheme is illustrated in Figure 13 for a small 2-ary 3-tree. There is one figure for each of the upward and downward channels in the normal and the two re-routing layers.

We will now proceed with the proof of deadlock freedom for the routing algorithm $R^{switch}_{deterministic}$. The proof follows the same method as for link faults, we show that there are no transitions from any link to another link with a lower sequence number.

Theorem 3: $R^{switch}_{deterministic}$ is deadlock-free.

Proof: All channels in the network are given a sequence number according to the above rules. The routing algorithm is deadlock-free if we can show that it is impossible to move to a channel that has a sequence number lower than or equal to the current channel.

There are six specific situations for any packet forwarded in the network.

1) *The packet is forwarded upwards in NL*
   According to rule 1, all upward channels at the next link tier upwards have a larger sequence number than the current channel.
2) *The packet is forwarded downwards in NL*
   According to rules 2 and 3, all downward channels at the next link tier downwards have a larger sequence number than the current channel, regardless of whether the current channel is a normal channel or a M1 channel.
3) *The packet is forwarded upwards in M1*
   This only takes place in the downward routing phase. According to rule 4, all upward ML1 channels at the current link tier have a larger sequence number than any downward normal channel at the same tier, and all re-routing channels at the tiers above. In addition, the next re-routing channel always has a larger sequence number than the current re-routing channel, following the ordered sequence D and rule 4.
4) *The packet is forwarded downwards in M1*
   There are three possibilities. Either the packet is returning to the U-turn switch, it is on a fault-free downward path towards the destination, or it must be re-routed again because the shortest downward path contains another fault. We call the current switch z. z has the same position in the re-routing sequence D for all U-turn switches in the switch group. Thus all upward M1 channels in the switch group connected to z will have the same sequence number, as will all the downward M1 channels in the switch group connected to z (rule 4, see Figure 13 for an example). The sequence number of the downward M1 channel from z, whether it returns back to the U-turn switch, proceeds towards the destination, or is re-routed again, must therefore be larger than the sequence number of any upward M1 channel that may have led a packet to z and any downward M2 channel that may have led a packet to z, since a downward M1 channel has a sequence number that is 3 larger than the same upward M1 channel (rule 4), and any M2 channel at the tier above connected to the same switch has a sequence number at most two larger than the upward M1 channel (rule 5).
5) *The packet is forwarded upwards in M2*
   This only takes place after an upward M1 channel. Following from Corollary 1all upward M1 channels connected to the downward ports of the same switch have the same sequence number. Hence, following rule 5, the upward M2 channel from this switch will have a sequence number one larger than any upward M1 channel leading to it.
6) *The packet is forwarded downwards in M2*
   There are two possibilities. Either the packet must return to the U-turn switch, or it has a fault-free downward path towards the destination. We call the current switch z. Following from Corollary 2 z is connected through downward links only to switches with the same sequence number in D. Consequently, following rule 5, all upward M2 channels to z have the same sequence number, which is one larger than any upward M1 channel that may have preceded it, and all downward M2 channels from z have the same sequence number, which is one larger than any upward M2 channel that may have preceded it.

No packet forwarded in the network will ever reach a channel that has a sequence number that is lower than or equal to that of any channels it has previously traversed, and $R^{switch}_{deterministic}$ is deadlock-free. ■

As is the case for link faults, resuming the use of a previously failed switch once it is restored to service is also deadlock-free, because the numbering scheme that we have utilised here remains intact.

F. Livelock freedom and connectivity for $R^{link}_{adaptive}$

We now prove the connectedness of $R^{link}_{adaptive}$.

Lemma 9: $R^{link}_{adaptive}$ is connected and livelock free when there are fewer than k link faults in every switch group.

Proof: Lemma 2 guarantees that there exists a path from every switch to every destination with less than $k-1$ arbitrary link faults. It is therefore sufficient to show that the algorithm

is livelock free. The number of links is finite; hence, a livelock requires that a packet is forwarded in a loop. There must therefore exist a set of switches that the packet may traverse 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 re-routing around a fault. However, the re-routing vector in point 4 of the algorithm and the re-routing in point 2b guarantee that all the upper-tier switches in the switch group containing the faults are used as next hops at most once; hence, none of these can be involved in a livelock. Given that there are fewer than k link faults, it is guaranteed that there is a path that moves the packet out of the switch group and one tier lower down towards the destination. Consequently, every time a packet is re-routed, it will find a path that brings it one stage closer to its destination, and the algorithm is livelock free.

■

1) Livelock freedom and connectivity for $R^{switch}_{adaptive}$: The proof of connectivity and livelock freedom for $R^{switch}_{adaptive}$ is quite similar to that for $R^{link}_{adaptive}$.

Lemma 10: $R^{switch}_{adaptive}$ is connected and livelock free with less than $k-1$ arbitrary link and switch faults.

Proof: Lemma 4 guarantees that there exists a path from every switch to every destination with less than $k-1$ arbitrary link and switch faults. It is therefore sufficient to show that the algorithm is livelock free. The number of links is finite; hence, a livelock requires that a packet is forwarded in a loop. There must therefore exist a set of switches that the packet may traverse 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 re-routing around a fault. However, the *port*-field utilised in points 3 and 5 guarantees that the re-routed packet always returns to the same U-turn switch until it has progressed one tier closer to the destination. Furthermore, the re-routing vector in point 4 of the algorithm, the port field utilised in points 4 and 5, and the re-routing in point 2b guarantees that all the upper-tier switches in the switch group containing the faults are used as next hops at most once; hence, none of these can be involved in a livelock. Given that there are fewer than k switch faults, it is guaranteed that there is a path that moves the packet out of the switch group and one tier lower down towards the destination. Consequently, every time a packet is re-routed, it will find a path that brings it one tier closer to its destination, and the algorithm is livelock free. ■

We have shown above that $R^{link}_{adaptive}$ and $R^{switch}_{adaptive}$ is connected when there are fewer than k link faults in the fat-tree. We now show that freedom from deadlock only can be guaranteed within the same number of link faults.

G. Deadlock Freedom for $R^{link/switch}_{adaptive}$

Definition 19: The *subtree rooted at switch s* consists of all the links and switches reachable in the downward direction from s. The *subtree leaves* are the processing nodes connected to the bottom tier switches of the subtree.

The fat-tree is made up of multiple subtrees at every level of the fat-tree. Every switch at a tier l above the bottom tier ($l < n-1$) is the root of a subtree. Forwarding in a fault-free fat-tree takes place in the subtree of the least common ancestor chosen in the upward phase.

Every switch is a member of a specific set of subtrees, i.e. one for each switch that can be reached in the upward direction from that switch. A *top-rooted subtree* is a subtree of a top tier switch. An example of a top-rooted subtree is shown in Figure 14 on page 26.

Definition 20: Two switches are *members of the same set of top-rooted subtrees* if they have all top-tier subtree roots in common.

Definition 21: A *subtree that is fault-free above tier l* is a subtree whose root is at the top tier of the fat-tree (tier 0), and whose leaves are the processing nodes. All the links and switches above tier l in the tree are fault-free.

In a subtree that is fault-free above tier l, all switches in the top-rooted subtree at tiers $\{0, \ldots, l-1\}$, and all links going downwards from the switches, are fault free.

Definition 22: A *subset of leaves* consists of all the leaves reachable through a single downward link from a subtree root.

The numerous U-turns caused by packets being re-routed around faults may close cycles in the dependency graph, and thus potentially cause deadlock. We now show that the cycles do not cause deadlocks, provided that the number of link faults does not exceed a certain threshold. We assume the use of virtual cut-through switches and we focus on the dependencies between the switch queues where the packets are buffered.

In a deterministically routed network, a necessary and sufficient condition for deadlock freedom is that the channel dependency graph is free from cycles. However, in an adaptive network, each packet may have several alternative next-hop queues. Which of these queues is chosen is determined by a selection function, which may depend on several factors, such as queue length and mean queuing time. As long as one of the alternative next-hop queues has available space, the packet may be forwarded to this queue. It follows that an adaptive network is not necessarily deadlocked even if there exists a cycle in the dependency graph. Even if there is a set of queues that form a dependency cycle, packets in the queues that form the cycle may have additional next-hop queues that are not part of the cycle, because of the adaptivity.

In [6] there is a comprehensive theory of deadlocks in adaptive cut-through networks. For completeness, we rephrase the part of this theory that is necessary for proving that our method provides freedom from deadlock.

Definition 23: A routing subfunction R1 of the routing function R* is a routing function that is defined on the same domain as R* but that provides a subset of the queues provided by R*.

R1 may thus 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. There may exist many routing subfunctions of R*. However, when we later apply the theory to our routing method, we will concentrate on one particular carefully chosen one.

For a network to be deadlocked, the network must have reached a state in which there is an assignment of packets to queues that prohibits the network from functioning. Now we

turn our attention to the precise nature of deadlock:

Packets that occupy space in one queue and request access to another form a relationship between the two queues in which the blocking of one may lead to the blocking of the other. We call this relationship a dependency.

Definition 24: There exists a *dependency* from one queue, a, to another, b, for the routing function R when there exists a legal configuration relative to R where packets in a may request b as the next hop according to R.

Definition 25: Let R1 be a routing subfunction of R*. There exists a *cross dependency* from queue a to queue b if there exists a legal configuration relative to R*, in which a packet in a may request b according to R1.

The difference between a dependency and a cross dependency is subtle, but important for our proof. If R1 is a routing subfunction of R*, then R1 has its own set of dependencies according to Definition 24. However, because R1 is a routing subfunction, an additional set of dependencies emerges from the interplay between R* and R1. In particular, R* may route some packets to destinations where R1 would not have routed them. Still, R1 may (and actually should, as becomes clear later) provide routing alternatives for these packets. These routing alternatives that R1 provides for packets that are "misplaced" relative to R1 give rise to cross dependencies.

Definition 26: Let R1 be a routing subfunction of R*. The *extended dependency graph* of R1 is a directed graph whose vertices are queues supplied by R1, and there is an arc from queue a to queue b if there is either a dependency or a cross dependency from a to b.

Definition 27: A routing function R is *connected* if R establishes a path to the packet destination for every packet in every queue in every legal configuration.

An important special case of the above definition is the case in which R1 is a routing subfunction of R*, and R* is the routing function that is operative in the network. In that case, for the routing subfunction R1 to be connected, it must establish paths for every configuration that is legal relative to R*. Finally, we present the main theorem of the theory in [6]. We make use of this theorem to show that our routing algorithm is free from deadlock:

Theorem 4: A routing algorithm R* is deadlock-free iff there exists a connected routing subfunction R1 of R* that has no cycles in its extended dependency graph.

To use Theorem 4, we must construct a routing subfunction that is connected and cycle-free in its extended dependency graph. In order to do this, we must first make some observations concerning the structure of the fat-tree.

Observation 1: For any top-tier switch (tier 0), there is only a single downward path between it and an arbitrary switch within its subtree.

Lemma 11: All upward links from a switch belong to disjoint sets of subtrees.

Proof: Assume that two of the upward links of a switch s have at least one subtree in common. In that case, there are at least two different paths from the root of the subtree down to s, and we have a contradiction with observation 1. ■

Lemma 12: When there are fewer than k faults in the network, every possible U-turn switch is connected to at least one subtree that is fault-free above its own tier.

Proof: The set of possible U-turn switches contains all switches in the fat-tree except the switches at tier zero. From lemma 11, we know that every possible U-turn switch is connected to k disjoint sets of subtrees, one set through each of its k upward links. As the sets are disjoint, a single fault may be in at most one of the sets. Therefore, $k-1$ is an insufficient number of faults to place one fault in each of the disjoint subtree sets. Thus, at least one of the upward links from any U-turn switch is connected to at least one fault-free subtree above its own tier. ■

These are the foundations necessary to construct our routing subfunction. Recall that this routing subfunction is not supposed to be implemented. It is defined only for proof purposes, because its mere existence proves that the fault-tolerant routing algorithm $R_{adaptive}^{link/switch}$ is deadlock-free. The steps of the routing subfunction $R'_{adaptive}$ of $R_{adaptive}^{link/switch}$ are listed below. We can use the same routing subfunction $R'_{adaptive}$ for both link and switch faults, the only difference is whether we re-route down one or two tiers.

- Choose a fault-free subtree above every U-turn switch.
- Do the following in the upwards direction until a least common ancestor of the source and destination has been reached:
  - Outside the chosen fault-free subtrees, adaptively forward the packet upwards.
  - Within a chosen fault-free subtree above a U-turn switch, route the packet upwards within this tree. If there is a conflict because a switch is part of two different chosen fault-free subtrees (above two different U-turn switches), route within the chosen subtree above a U-turn switch at the lowest level in the fat-tree.
- In the downwards direction, do the following:
  - Route the packet deterministically downwards towards its destination.
  - If a fault is encountered in the downward direction, re-route the packet one or two steps downwards for link or switch fault tolerance.
  - Forward the packet back up the chosen subtree that is fault-free.

The only limitation on $R'_{adaptive}$ relative to $R_{adaptive}^{link/switch}$ is which upward path may be taken for packets residing in a chosen subtree that is fault-free above a U-turn switch. The path of a packet following this routing subfunction is illustrated in Figure 14. One of the original paths from source to destination was $Source \rightarrow A \rightarrow B \rightarrow E \rightarrow Destination$. Once the link $B \rightarrow E$ fails, packets are forwarded in two tiers. First, a fault-free subtree at tier 1 that encompasses the three possible U-turn switches A, F, and G comes into existence. Second, all packets in A and B are forwarded towards the faulty link, and as they encounter it they are re-routed to one of the U-turn switches. From here on the packet enters the fault free subtree and follows this all the way to the destination via D and E. All subsequent packets from the source via A will have entered the fault-free subtree and are therefore required to continue following the tree to the destination from tier 1

and upwards (to tier 0). The path for all subsequent packets is therefore $Source \rightarrow A \rightarrow D \rightarrow E \rightarrow Destination$. Note that there are two more fault-free subtrees that have not been marked in the figure.

The U-turn switch will only be traversed by the first few packets encountering the link fault. This is because all subsequent packets will have to enter the fault-free subtree in the upward phase in order to reach the fault, but they will never encounter it because the packets are not allowed to leave the fault-free subtree. This obviously eliminates any possible cycles, because all packets that would otherwise have encountered the link fault follow the fault-free subtree.
Before we proceed with the main theorem, we prove a lemma on the nature of the dependencies of $R'_{adaptive}$.

Lemma 13: Let s be a U-turn switch at level l. In the extended dependency graph of $R'_{adaptive}$, there are no dependencies at or above level l that go out of the chosen fault-free subtree above s.

Proof:

Assume that the lemma does not hold. This means that there must be a possible situation in which a packet that resides at or above level l in the chosen fault-free subtree above s may be routed out of the subtree by $R'_{adaptive}$. This may be a packet routed there by $R'_{adaptive}$ (which will give rise to a dependency out of the subtree), or one routed there by $R^{link/switch}_{adaptive}$ (which will give rise to a cross dependency out of the subtree).

It is impossible to route out of a subtree in the downwards direction. Furthermore, since the subtree is fault-free down to level l, there are no U-turn switches at or above level l within the subtree. Therefore, our packet must be in its upwards phase. It is easy to observe that $R'_{adaptive}$ in the upward phase will route any packet within the fault-free subtree above s. Thus, the assumed situation cannot occur and the lemma holds. ■

We are now in a position to state our main theorem:

Theorem 5: When there are fewer than k faults in the network, the routing subfunction $R'_{adaptive}$ is connected and has no cycles in its extended dependency graph.

Proof: Assume there is a cycle of dependencies and cross dependencies in the extended dependency graph of $R'_{adaptive}$. Consider one of the U-turns in the cycle that is located closest to the bottom of the tree i.e. a U-turn that is such that there is no U-turn switch in the cycle at a switch at a lower tier. Let this U-turn be in switch s at tier l. Following the chain of (cross) dependencies that start from this U-turn, there is no (cross) dependency out of the subtree that is fault-free above tier l, according to lemma 13. The dependency chain will therefore remain within the fault-free subtree until it reaches tier l in the downward direction. For there to be a cycle, there must be another U-turn that takes the cycle back in the upward direction. The next U-turn in the cycle must, because of Lemma 12, be at a tier below l, which contradicts the way in which the first U-turn was chosen. Therefore, there cannot be any cycle. See Figure 14 to see how a re-routed packet cannot encounter any U-turns at any tier at or above tier l.

Finally, we show that $R'_{adaptive}$ is connected. In the upward phase, any packet is guaranteed to be able reach a top-tier switch (tier 0), and therefore also a least common ancestor. In the downward phase, connectedness is trivial when there are no faults. If the packet reaches a fault, the packet will be able to reach a U-turn switch as long as there are fewer than k faults. Let l be the tier of this U-turn switch. The packet will now be forwarded within the fault-free subtree above l, and thereby not reach another fault before it has reached level l on its way down again. But then the next U-turn switch will be at a level below l. Since there are not infinitely many levels, connectedness is proven by well-founded induction on the number of U-turn switches that the packet has to traverse. ■

Since $R'_{adaptive}$ is a routing subfunction of $R^{link/switch}_{adaptive}$ and we have shown that $R'_{adaptive}$ has no cycles in its extended dependency graph, we refer to Theorem 4 and conclude that $R^{link/switch}_{adaptive}$ is deadlock-free when there are fewer than k arbitrary link and switch faults respectively.

REFERENCES

[1] InfiniBand Trade Association. *InfiniBand Architecture. Specification Volume 1. Release 1.0a*. Available at http://www.infinibandta.com, 2001.

[2] A. Bermudez, R. Casado, F. J. Quiles, and J. Duato. Handling topology changes in infiniband. *IEEE Transactions on Parallel and Distributed Systems*, 18(2):172–185, 2007.

[3] S. Chalasani, C.S. Raghavendra, and A. Varma. Fault-tolerant routing in MIN based supercomputers. In *Supercomputing '90: Proceedings of the 1990 conference on Supercomputing*, pages 244–253. IEEE Computer Society Press, 1990.

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

[5] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[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] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato. Deterministic versus adaptive routing in fat-trees. In *Workshop on Communication Architecture on Clusters, as a part of IPDPS'07*, March 2007.

[8] 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.

[9] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[10] 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.

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

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

[13] R. Ponnusamy, A. Choudhary, and G. Fox. Communication overhead on the cm5: an experimental performanceevaluation. *Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 108–115, October 1992.

[14] I. D. Scherson and C.-K. Chien. Least common ancestor networks. In *International Parallel Processing Symposium*, pages 507–513, 1993.

[15] F. O. Sem-Jacobsen. *Towards a Unified Interconnect Architecture: Combining Dynamic Fault Tolerance with Quality of Service, Community Separation, and Power Saving*. Ph.d. thesis, University of Oslo, 2008.

[16] F. O. Sem-Jacobsen, O. Lysne, and T. Skeie. Combining source routing and dynamic fault tolerance. In Rajkumar Buyya Alberto F. De Souza and Wagner Meira Jr., editors, *Proceedings of The 18'th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 151–158, Washington, DC, USA, 2006. IEEE Computer Society.

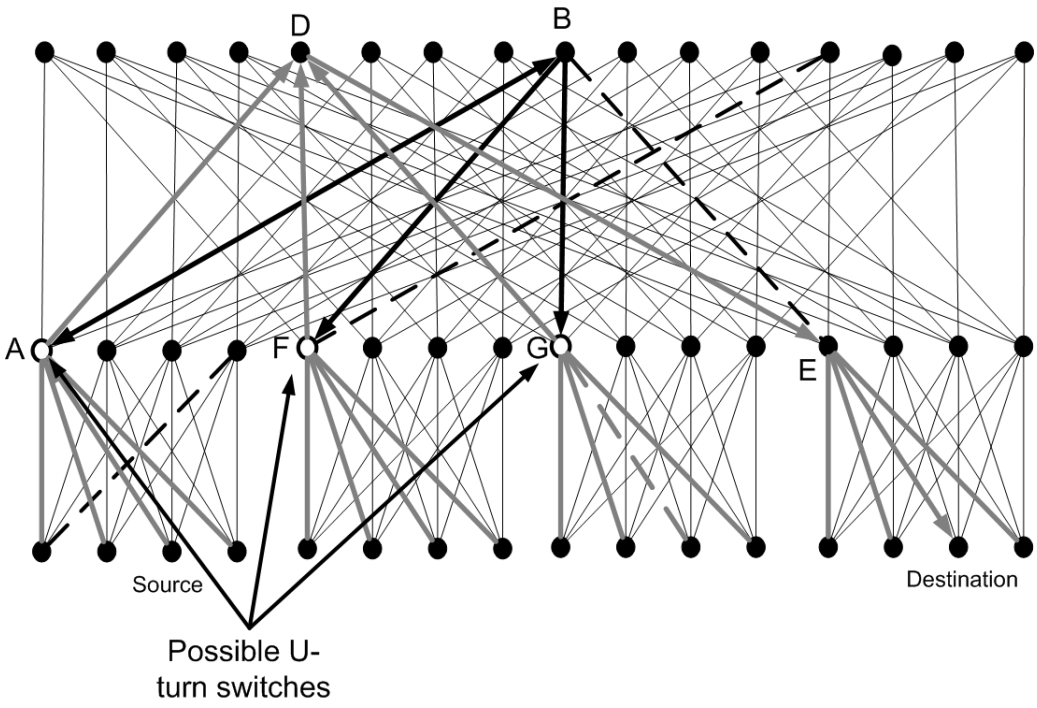


Figure 14. A fault-free subtree.

The bold, grey, unbroken links designate the fault-free subtree for the marked U-turn switch. The dashed links designate faults. The possible paths of a packet from source to destination are marked by the bold links, with arrows indicating the direction of the packet's travel.

[17] F. O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato. Dynamic fault tolerance in fat-trees. Technical report, Simula Research Laboratory, 2009.

[18] 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.

[19] J. Sengupta and P.K. 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.

[20] J. Sengupta and P.K. Bansal. High speed dynamic fault-tolerance. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology, 2001. TENCON.*, volume 2, pages 669–675, 2001.

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

[22] T. Skeie. A fault-tolerant method for wormhole multistage networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 637–644, 1998.

[23] I. T. Theiss and O. Lysne. Froots, a fault tolerant and topology agnostic routing technique. *IEEE Transactions on Parallel and Distributed Systems*, 2005.

[24] Top 500 supercomputer sites. http://www.top500.org/, November 2009.

[25] H.-Y. Tyan. *Design, Realization and Evaluation of a Component-Based Compositional Software Architecture for Network Simulation*. PhD thesis, Ohio State University, 2002.

[26] N.-F. Tzeng, P.-C. Yew, and C.-Q. Zhu. A fault-tolerant scheme for multistage interconnection networks. In *ISCA '85: Proceedings of the 12th annual international symposium on Computer architecture*, pages 368–375. IEEE Computer Society Press, 1985.

[27] 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.

[28] M. Valerio, L.E. Moser, and P.M 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.

[29] M. Woodacre, D. Robb, D. Roe, and K. Feind. The sgi altix tm 3000 global shared-memory architecture. *SGI HPC White Papers*, 2003.