Deterministic versus Adaptive Routing in Fat-Trees *

C. Gómez, F. Gilabert, M.E. Gómez, P. López and J. Duato

Dept. of Computer Engineering
Universidad Politécnica de Valencia
Camino de Vera, 14, 46071–Valencia, Spain
{crigore, fragivil}@gap.upv.es and {megomez, plopez, jduato}@disca.upv.es

Abstract

Clusters of PCs have become very popular to build high performance computers. These machines use commodity PCs linked by a high speed interconnect. Routing is one of the most important design issues of interconnection networks. Adaptive routing usually better balances network traffic, thus allowing the network to obtain a higher throughput. However, adaptive routing introduces out-of-order packet delivery, which is unacceptable for some applications. Concerning topology, most of the commercially available interconnects are based on fat-tree. Fat-trees offer a rich connectivity among nodes, making possible to obtain paths between all source-destination pairs that do not share any link. We exploit this idea to propose a deterministic routing algorithm for fat-trees, comparing it with adaptive routing in several workloads. The results show that deterministic routing can achieve a similar, and in some scenarios higher, level of performance than adaptive routing, while providing in-order packet delivery.

1 Introduction

In large parallel computers, high-performance interconnection networks are crucial to achieve the maximum performance. Routing is a critical design issues of interconnection networks [4]. The routing strategy determines the path that each packet follows between a source–destination pair. In deterministic routing schemes, an injected packet traverses a fixed, predetermined path between source and destination, while in adaptive routing schemes the packet may traverse a number of alternative paths. Deterministic routing algorithms usually do a very poor job balancing traffic among the network links, but they are usually easier to implement and easier to be deadlock-free. Moreover, for networks in which the ordering of messages between particular source–destination pairs is important, deterministic routing is often a simple way to guarantee in-order delivery. This is the case, for example, for certain cache coherence protocols and some communication libraries. On the other hand, adaptive routing algorithms take into account the status of the network in order to make the routing decisions. This information may include the status of links or the queue lengths. Adaptive routing better balances network traffic, thus allowing the network to obtain a higher throughput. A good adaptive routing algorithm should outperform a deterministic one, since it uses network state information that is not available for deterministic routing.

Cluster-based machines use any of the commercial high-performance switch-based point-to-point interconnects. Either regular direct networks (tori and meshes) or indirect multistage networks (MINs) are the usual choice. In particular, fat-trees have raised in popularity in the past few years (i.e., Myrinet [11], InfiniBand [8], Quadrics [12]).

The authors in [1] compare an oblivious routing algorithm with an adaptive routing algorithm for multistage networks, and conclude that an adaptive algorithm achieves a higher performance. The readers should take into account that oblivious routing is not the same than deterministic routing [4, 3], since oblivious routing can provide several paths for a source–destination pair and the routing decision is taken without considering (oblivious to) network status.

In this paper, we focus on routing in fat-trees. In particular, we propose a deterministic routing algorithm for fat-trees. As stated above, deterministic routing has some advantages over adaptive routing, for instance simplicity or in–order packet delivery. We will show that the proposed deterministic routing can be implemented in a compact-way and that it is able to obtain similar or even better perfor-

*This work was supported by the Spanish MCYT under Grant TIN2006-15516-C04-01, by CONSOLIDER-INGENIO 2010 under Grant CSD2006-00046 and by the European Commission in the context of the SCALA integrated project #27648 (FP6).

mance than adaptive routing in fat-trees. The rest of the paper is organized as follows. Section 2 revises the fat-tree topology and presents the notation and assumptions used in the following sections. Section 3 describes an adaptive routing algorithm for fat-trees and a possible implementation using Interval Routing (IR) [2]. Section 4 present the proposed deterministic routing algorithm for fat-trees and shows how it can be implemented using Flexible Interval Routing (FIR) [7]. Section 5 evaluates its performance. Finally, some conclusions are drawn.

2 Fat-Tree Topology

The k-ary n-trees are a parametric family of regular multistage topologies. The number of stages is n and k is the arity or the number of links of a switch that connect to the previous or to the next stage (i.e., the switch degree is $2k$). A k-ary n-tree is able to connect $N = k^n$ processing nodes using nk^{n-1} switches.

Each processing node is represented as a n-tuple $\{0, 1, ..., k-1\}^n$, and each switch is defined as a pair $\langle s, o \rangle$, where s is the stage where the switch is located at, $s \in \{0..n-1\}$, and o is a $(n-1)$-tuple $\{0, 1, ..., k-1\}^{n-1}$ which identifies the switch inside the stage. Figure 1 shows a 2-ary 4-tree, with 16 processing nodes and 32 switches.

In a fat-tree, two switches $\langle s, o_{n-2}, ..., o_1, o_0 \rangle$ and $\langle s', o'_{n-2}, ... , o'_1, o'_0 \rangle$ are connected by an edge if $s' = s+1$ and $o_i = o'_i$ for all $i \neq s$. On the other hand, there is a edge between the switch $\langle 0, o_{n-2}, ..., o_1, o_0 \rangle$ and the processing node $p_{n-1}, ..., p_1, p_0$ if $o_i = p_{i+1}$ for all $i \in \{n-2, ..., 1, 0\}$. This edge is labeled with p_0 in the stage 0. In what follows, we will assume that descending links are labeled from 0 to $k-1$, and ascending links from k to $2k-1$.

3 Adaptive Routing in Fat–trees

In k-ary n-trees, minimal routing from a source to a destination can be accomplished by sending packets upwards to one of the nearest common ancestors of the source and destination nodes and then, from there, downwards to destination. When crossing stages in the upwards direction, several paths are possible, thus providing adaptive routing. In fact, each switch can select any of its up output ports. Once a nearest common ancestor has been reached, then the packet is turned around and sent downwards to its destination and just a single path is available.

The stage up to which the packet must be forwarded is obtained by comparing the source and destination components beginning from the most significant one. The first pair of components that differs indicates the last stage to forward up the packet. For instance, in order to send a packet from node $p_{n-1}, ..., p_1, p_0$ to node $p'_{n-1}, ..., p'_1, p'_0$, the packet must be sent up to the stage i, if $p_j = p'_j$ for $j \in \{n-1..i+1\}$ and $p_i \neq p'_i$. Once in the stage i, the descending path is deterministic. At each stage, the descending link to choose is indicated by the component corresponding to that stage in the destination n-tuple. In the example, from stage i, the packet must be forwarded through the p'_i link; from stage $i-1$ through link p'_{i-1}, and so on.

3.1 Adaptive Routing Implementation

This routing algorithm can be easily implemented using Interval Routing (IR) [2]. In IR, each switch output port has one associated interval. Each packet is forwarded through the output port whose interval contains the destination of the packet. The interval associated to each output port can be cyclic and is implemented with two registers. We will refer to these two registers as First Interval (FI) and Last Interval (LI). Moreover, this scheme requires a simple hardware, at most a pair of comparators for each output link, therefore it is also very fast.

Figure 1 presents an example of configuration of the IR registers for a 16-node 2-ary 4-tree. As it can be seen, the interval associated to some output ports must be cyclic. As an example, we describe how to route with IR a packet from node 1 to node 4. Switch 0 can use both ascending links (links 2 and 3) to route the packet, since destination 4 is included in the intervals associated to both links. Assume that the selection function selects link 3, so switch 9 is reached. Switch 9 can route the packet through any of its ascending links. Assume link 2 is selected, then the packet reaches switch 17. At switch 17, only link 1 is allowed to route the packet destined to node 4, so the packet arrives to switch 11. At this switch, only link 0 is allowed to route the packet, and switch 2 is reached. Finally switch 2 delivers the packet to node 4 through link 0.

Next, we present a general algorithm to fill the FI and LI registers to support adaptive routing in fat-trees. Figure 2 shows the prototyped FI and LI configuration for the links of a generic switch of a k-ary n-tree. The switch is labeled as $\langle s, o_{n-2}, o_{n-3}, ..., o_1, o_0 \rangle$, so it is located at the stage s. First, we identify the destinations reachable through the descending links, and later the rest of destinations reachable through all the ascending links.

The $\langle p_{n-1}, ..., p_1, p_0 \rangle$ nodes that are reachable by the descending links can be easily computed from the switch components. In particular, $p_i = o_{i-1}$ for $i \in \{n-1, .., s+1\}$. This set of destinations is split in several subsets that can be reached from each descending link depending on the p_s component, being s the switch stage. The subset of nodes whose $p_s = 0$ are reachable through link 0, the subset of destinations whose $p_s = 1$ are reachable through link 1, and so on. As an example, link 0 of switch $\langle s, o_{n-2}, ..o_1, o_0 \rangle$ forwards packets destined to nodes $\langle o_{n-2}, ..., o_s, 0, X...X \rangle$, that is, $FI_{desc.} = \langle o_{n-2}, ..., o_s, 0, 0...0 \rangle$ and $LI_{desc.} = \langle o_{n-2}, ..., o_s, 0, k-1...k-1 \rangle$.

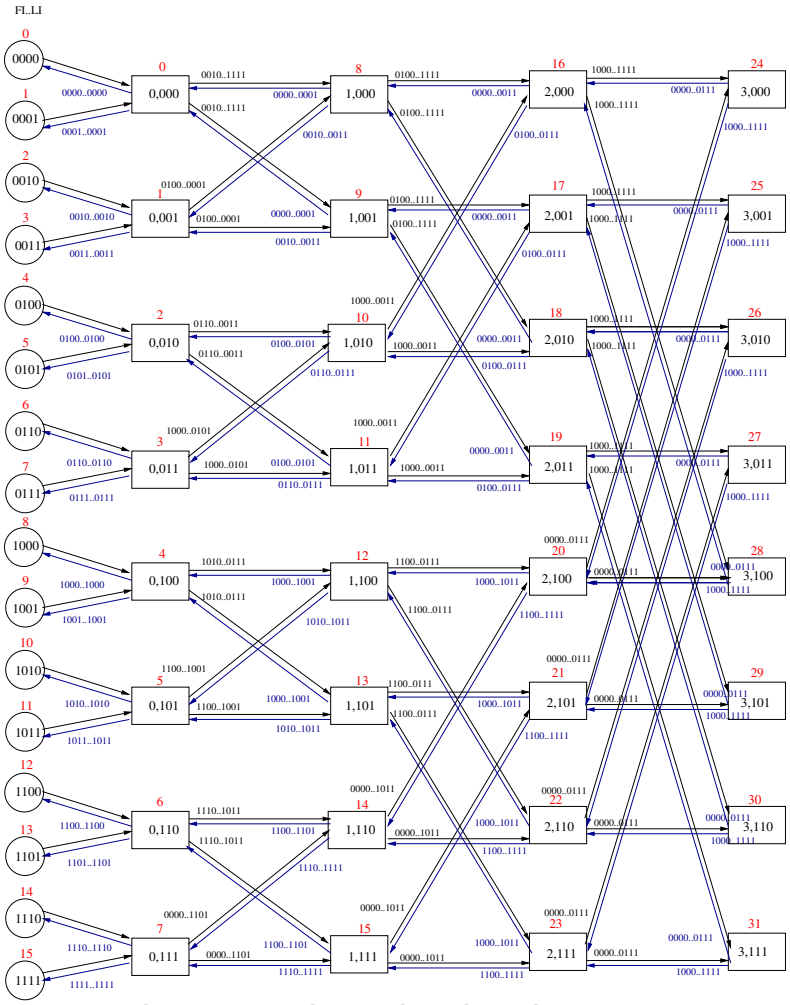

Figure 1. Adaptive routing with IR in a 2-ary 4-tree.

The destinations that are not reachable through the descending links, but are reachable through the ascending links, are the ones that do not meet $p_i = o_{i-1}$ for $i \in \{n-1, .., s+1\}$. Indeed, this set is reachable through all the ascending links. The FI register of the ascending links must store the next destination to the largest one reachable through the descending links. That is, the next destination to the one stored in the LI register of the $k-1$ descending link. Likewise, the LI register of the ascending links must store the previous destination to the smallest one reachable through the descending links. That is, $FI_{asc.} = (LI_{link_{k-1}} + 1) \bmod k^n$ and $LI_{asc.} = (FI_{link_0} - 1 + k^n) \bmod k^n$, being k^n the number of nodes in the network[1]. This interval is valid for all the ascending links of the switch and can result in a cyclic interval.

[1]For $LI_{asc.}$ we add k^n in order avoid setting a negative destination value.

4 Deterministic Routing in Fat–trees

In this section, we propose a deterministic routing algorithm for fat-trees. Our challenge is to propose an efficient mechanism to reduce the multiple ascending paths in a fat-tree into a single one for each source–destination pair. The path reduction should be done trying to balance network link utilization. All the links of a given stage should be used by a similar number of paths. This is easy to achieve in the ascending phase. A simple idea is to divide the adaptive up interval of a switch into k sub–intervals of the same size, but by using this mechanism the descending links are not balanced. We analyzed several approaches, trying to balance both routing phases, and found that a good alternative is to shuffle, at each switch, consecutive destinations in the ascending phase. In other words, consecutive destinations are distributed among the different ascending links, reaching different switches in the next stage.

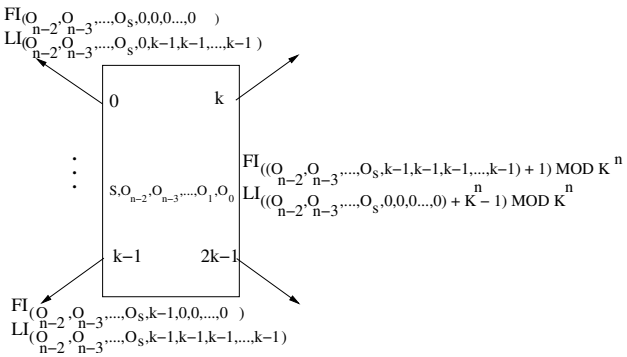

Figure 2. Prototyped register configuration.

We will explain the mechanism by using an example. Figure 3 shows the destination node distribution in the ascending and descending links of a 2-ary 3-tree using our proposal. In the figure, each ascending link has been labeled (in bold-italic) with the destinations whose packets will be forwarded through it. In the first stage, consecutive destinations are shuffled between the two up links. To do that, the least significant component of the packet destination address (the least significant bit) is used to select the ascending output port. That is, packets that must be forwarded upwards select the ascending output port indicated by the least significant component of the packet destination (p_0). Therefore, consecutive destinations are sent to different switches in the next stage. At the second stage, all the destinations that reach a switch have the same least significant component. Hence the component to consider in the selection of the up output port in this stage is the following one in the destination address. For instance, at switch 4, only packets destined to nodes 0, 2, 4 and 6 reach that switch and only packets destined to nodes 4 and 6 must be forwarded upwards. Packets destined to node 4 select the first up link, and packets destined to node 6 the other one.

Considering all the switches of stage 1, packets destined to nodes 0, 1, 4 and 5 use the first up output port of the switches and those packets destined to nodes 2, 3, 6 and 7 use the second output port. That is, the second least significant component of the packet destination is used. This mechanism distributes the traffic destined to different nodes, as shown in Figure 3. As it can be seen, packets destined to the same node reach the same switch at the last stage independently of their source node. Each switch of the last stage receives packets addressed only to two destinations, and packets destined to each one are forwarded through a different descending link.

The bottom of Figure 3 shows the number of paths (source–destination pairs) that use of each link at each stage. Both, the ascending and descending links of a given stage are used by the same number of paths. So, traffic in the network is completely balanced.

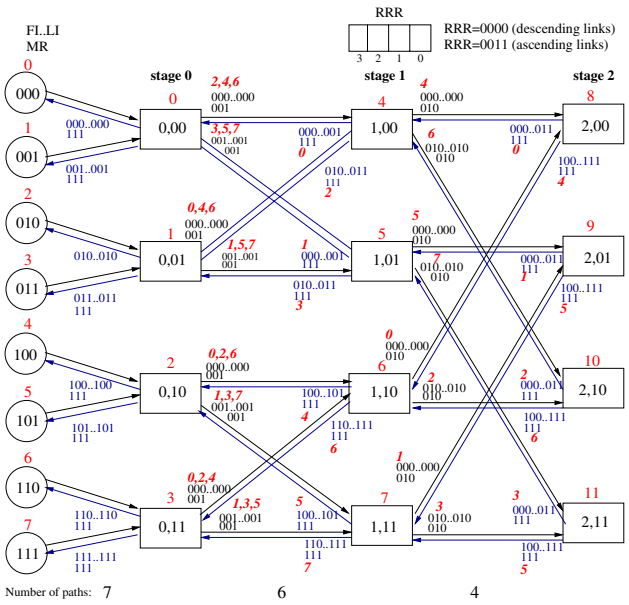

Figure 3. Deterministic routing in a 2-ary 3-tree with FIR registers.

4.1 Deterministic Routing Implementation

In this section, we show how the proposed deterministic routing strategy for fat-trees can be easily implemented using the Flexible Interval Routing (FIR) [7]. To make this paper self-contained, we first summarize FIR.

FIR can implement the most commonly-used routing algorithms in meshes and tori. In FIR, as in IR, each output port has also an associated cyclic interval, which is implemented with the FI and LI registers. But, in order to add flexibility, additional registers are associated to the output ports. In particular, each output port has a Mask Register (MR). This register indicates which bits of the packet destination address are compared with the output port bounds (provided by the FI and LI registers).

In order to guarantee deadlock freedom, some routing restrictions must be usually applied. These routing restrictions are taken into account in FIR by means of the Routing Restrictions Register (RRR), which defines, for each output port, which other output ports of the switch should be selected prior to this one. This register has one bit per output port. For a given output port i, the j bit in the RRR indicates whether the output port j has more preference (bit set to 1) or not (0) than output port i. Thus, the final routing decision for an output port i is obtained not only by comparing the masked destination with the interval bounds, but also by checking the bits in its RRR.

Now, we show how these registers can be configured to route packets in fat-trees following the proposed determi-

```
RRR={0}^K{1}^K
stage s
    (* selection of the bits corresponding to the current stage *)
    MR={0}^(n*r−(s+1)*r){1}^r{0}^(s*r)
    link L
        FI..LI={0}^(n*r−(s+1)*r)L−k{0}^(s*r) .. {0}^(n*r−(s+1)*r)L−k{0}^(s*r)
(* the bits corresponding to the current stage are given by the link identifier *)
(* they must be equal to L−k, being represented by r bits *)
being
   k = arity of the switch
   n = number of stages
   r = log(k); (* bits used by each stage in the destination addresses *)
   n*r = bits in destination identifiers
```

Figure 4. Register configuration in the ascending.

nistic routing algorithm. Notice that the adaptive routing algorithm described in section 3 can be also implemented with FIR, since IR is a subset of FIR.

We begin explaining how to configure the ascending links. They are configured in a very different way than in the adaptive case, since the number of ascending paths is reduced to one for each source–destination pair. At each switch, the ascending link to use is obtained from the packet destination component corresponding to the stage at which the switch is located. A given packet is only forwarded upwards through that up link. That is, at stage 0, the least significant component is used, at stage 1 the following one and so on. To obtain the proper component from the destination identifier corresponding to the switch stage, we use the Mask Register (MR). The MR of each ascending link sets to 1 the bits corresponding to the component associated to the switch stage. Figure 3 shows the FIR register configuration for a 2-ary 3-tree. In the first stage, MR is set to 001, as only the least significant bit is selected and compared with FI and LI. Packets are forwarded through the ascending output port depending on the least significant component of its destination. At stage 1, the next bit or component is considered (MR is set to 010) and so on. In k-ary n-trees with $k > 2$ the components have more than one bit and, thus, in the MR more than one bit is set to 1 to select the component given by the switch stage.

Descending links have the same values stored in FI and LI as the ones with adaptive routing, since the path reduction is only done in the upwards phase. As the MR is not used in the downwards phase, it is set to all 1s to select all the bits in the destination address. Notice that, the same downwards paths valid in the adaptive case are also valid in the deterministic case, but only one is actually used.

Notice that in Figure 3 the destinations reachable through the descending links of a switch (for instance, destinations 0 and 1 at switch 0) are also included in the ascending intervals, so packets destined to that nodes could be incorrectly forwarded through those upwards links. To avoid this problem, the RRR register is used to give preference to the descending links over the ascending ones and guarantee a minimal path. In the RRR, the half lowest significant bits correspond to the descending links and the half most significant to the ascending ones. Therefore, in the ascending links (links 2 and 3) the RRR stores 0011, to give preference to links 0 and 1. In this way, as an example, when routing a packet to destination 0 at switch 0, both output ports, 0 and 2, may be allowed, since this destination, after being masked, is included in the intervals associated to both output ports, but as output port 2 gives preference to output port 0, output port 2 is not finally returned.

Figure 4 shows an algorithm for configuring the FIR registers of the ascending links. The RRRs have the half least significant bits set to 1 and the half most significant bits set to 0, to give preference to the descending links. Notice that with $k \neq 2$, the MRs will have as many bits set to 1 as the bits needed to represent a component in the destination address, that is $log(k)$. These 1s will be located in the position corresponding to the switch stage and will select these bits in the destination identifiers. The rest of the MR will be set to 0. On the other hand, the FI and LI registers of an ascending link will select the destinations that have the identifier of the ascending link in the position of the component given by the switch stage (s in Figure 4). Since ascending links are labeled from k to $2k - 1$, the value in the destination identifier component will be $L - k$, L being the link identifier. The configuration for the descending links is not shown because it is very simple. The FI and LI registers are the same as the adaptive case, and the MR is set all to 1s to select all the bits in the destination address for being compared with FI and LI. The RRR is set to all 0s, since no preference is given to the other output ports.

5 Performance Evaluation

5.1 Network Model

To evaluate the routing algorithm proposed above, a detailed event-driven simulator has been implemented. The simulator models a k-ary n-tree with FIR routing and virtual cut-through switching. Each router has a full crossbar with queues both at the input and output ports. We assumed that it takes 20 clock cycles to apply the routing algorithm; switch and link bandwidth has been assumed to be one flit per clock cycle; and fly time has been assumed to be 8 clock cycles. These values were used to model Myrinet networks in [5]. Credits are used to implement the flow control mechanism. Also, each port link has a two-packet buffer.

When adaptive routing is used, a selection function must be applied after applying the routing function. Remember that FIR implements the routing function. In [1], the authors compare several selection functions for adaptive routing, but they only consider the peak throughput in the evaluation, so they conclude that selection function is not critical. However, in [6], several selection functions for fat-trees

were proposed and the authors notice that selection function has a high impact over average latency. For this, we only consider the two most representative selection functions proposed in [6]. The best one was **Static And Destination Priority (SADP)**. This selection function gives preference, among all the ascending links of a switch, to the same one obtained by applying the deterministic routing algorithm proposed in this paper. If this output port does not have free space, then another one is selected. The other selection function we show in this paper is **First Free (FF)**, which is the simplest, but also the worst one. It selects the first physical link which has free space.

We have evaluated the uniform traffic pattern, with and without hot-spot nodes, and I/O traces. In the first case, message destination is randomly chosen among all the processors. Packet size is 8 KB. The I/O traces used in our evaluations were provided by Hewlett-Packard Labs. They include I/O activity generated in the early 1999 at the disk interface of the $cello$ system. The $cello$ system is a timesharing system with a storage subsystem of twenty-three disks. They provide information both for the requests generated by the hosts and the answers generated by the disks. A detailed description of similar traces of 1992, collected in the same system, can be found in [13]. We will use packets with a payload equal to the size specified in the trace for the I/O accesses, but if the access is larger than 1 KB, we will split it into packets with, at most, a payload of 1 KB.

We have implemented a basic approach to ensure in-order delivery with the adaptive routing algorithm. It is similar to the one proposed in [10]. This approach uses a reorder-buffer (different buffer sizes have been used) at the destination node NIC to store out-of-order packets. Every time a packet is sent, its sequence number is included in its header. When a packet arrives out of order at the destination, it is stored in the reorder-buffer to wait for all packets with smaller sequence number. On the other hand, to prevent unnecessary packet retransmission, the source node does not inject packets if the destination buffer does not have enough free space to store all the packets that have been sent previously. This complicates the adaptive routing implementation, as a reorder-buffer is required at the destination and an end-to-end flow control is needed. Our reorder mechanism for adaptive algorithms is ideal, it has not any reordering cost (i.e., the acknowledge control messages do not consume network bandwidth). The difference in the results is produced only by the delay introduced by out-of-order arrived packets.

5.2 Evaluation Results

5.2.1 Synthetic Traffic

We have evaluated a wide range of k-ary n-tree topologies, k being 2, 4, 8, 16 and 32 and n being up to 8. with uniform pattern traffic, with and without hot-spot traffic. In the hot–spot traffic a percentage of traffic is sent to a single node, the rest of the traffic is uniformly distributed.

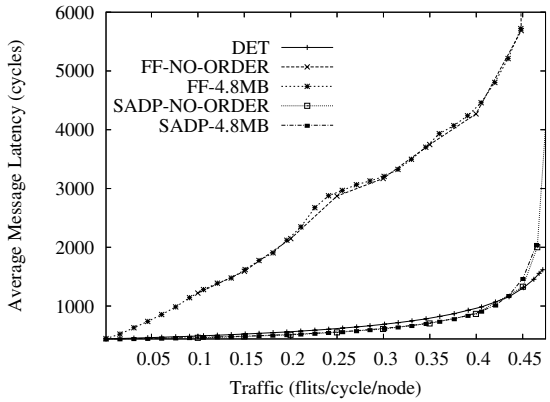

Figure 5. Average message latency versus accepted traffic for a 4-ary 4-tree with uniform traffic.

Figure 5 plots the average packet network latency versus the average accepted traffic for a 4-ary 4-tree with the uniform traffic pattern, without guaranteeing in-order packet delivery and guaranteeing in-order packet delivery. In deterministic routing in-order delivery is ensured by design. As it can be observed, the deterministic routing proposed in this paper strongly outperforms adaptive routing with the FF selection function and obtains roughly the same throughput than the adaptive routing algorithm when using the SADP selection function, which was the one that provided the best results. For the uniform traffic pattern, almost all the packets are delivered with the correct order, because traffic rate is distributed among all destinations. This is why similar performance results are obtained for all the different reorder-buffer sizes in the adaptive case. However, deterministic routing obtains a slightly higher latency than the adaptive one, because deterministic routing only can use one ascending link for a given packet, so it forces the packet to wait until it is available. On the other hand, near the saturation point, the network latency is smaller for the deterministic routing, because it classifies network traffic, thus reducing contention.

Figure 6 show the performance results for hot–spot traffic patterns for a 4-ary 4-tree. In Figure 6.(a), one random destination receives 5% of the traffic, while, in Figure 6.(b) one random destination receives 20% of the traffic. In the former case, deterministic routing obtains a slightly higher throughput than the adaptive ones when guaranteeing in-order delivery. In adaptive routing, when the preferred ascending link of a packet is not available, that packet is

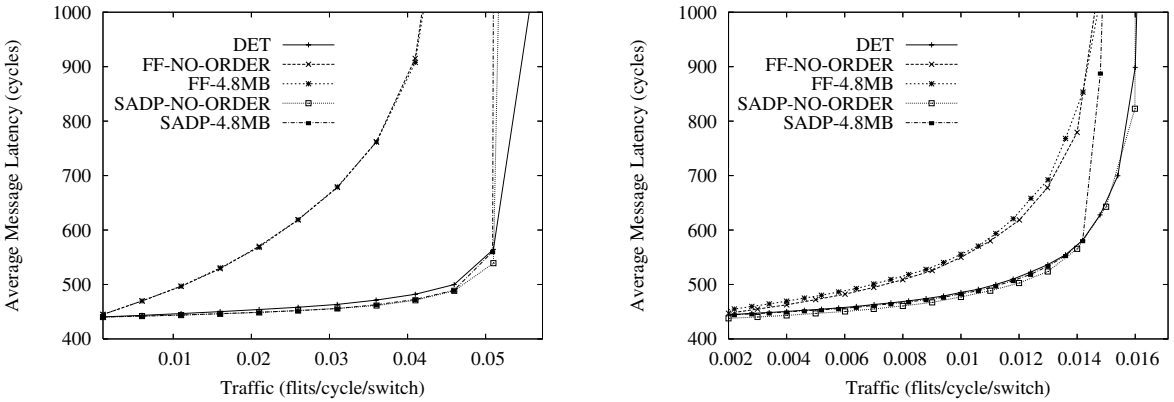

Figure 6. Average message latency versus accepted traffic for a 4-ary 4-tree with in-order delivery (a) hot–spot 5% (b) hot–spot 20%.

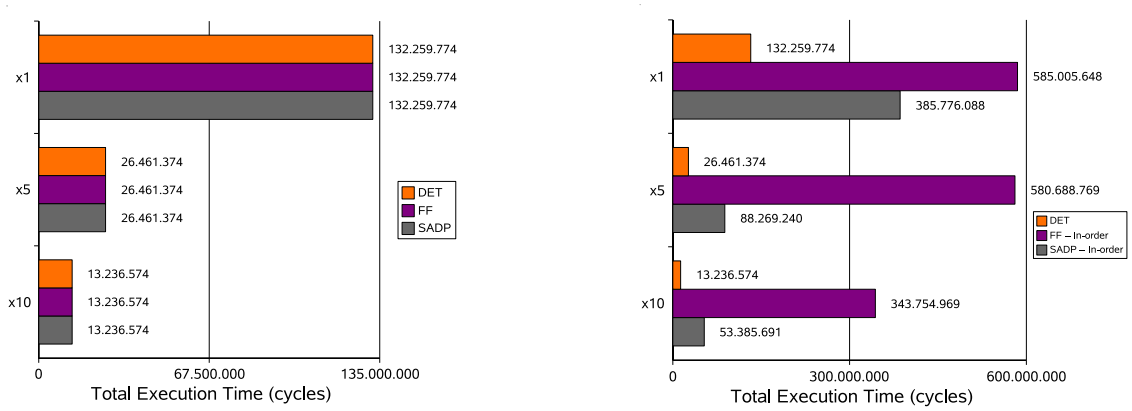

Figure 7. Total execution time for traces in a 2-ary 7-tree with different compression factors (1, 5 and 10) (a) without in-order delivery (b) with in-order delivery using a 8MB buffer size.

routed using another one, spreading the congestion to all the different alternative paths. However, with deterministic routing, when this ascending link has not buffer space left or is not available, that packet is stopped until it becomes available. So, in the deterministic routing algorithm, that congestion is not spread. The congestion only affects to the different paths that share links, and in the deterministic algorithm, the number of paths that share the same link is minimized. Figure 6 also shows the performance for a hot–spot traffic pattern when in-order delivery is not required. As it can be seen, the performance results obtained are similar to the ones obtained when in-order delivery is required, because the key issue here is not the number of out-of-order packets received, but the fast congestion of the network. But for SADP when a low buffer size is used the performance achieved by the network is fairly smaller. When the hot-spot destination node receives 20% of the traffic (Figure 6.(b)), all the routing algorithms obtain a very low performance. Anyway, deterministic is not worst than the adaptive ones.

Other traffic patterns have been also analyzed (like perfect shuffle and complement) and the overall results are qualitatively similar to the ones presented in this paper. The results for other networks sizes are also similar to the ones shown. Additionally, we have analyzed the worst traffic pattern that we can conceive for our proposed routing algorithm. In that traffic pattern, messages are only sent to processing nodes whose id has the same last component. For instance, in a 2-ary 2-tree, if all destinations have the last component equal to 0, the possible destinations should be 0 and 2. For this traffic pattern, the effective bandwidth is half the total bandwidth with deterministic routing (in the generic case, it would be $1/k$), because only the links labeled as 2 will be used in the ascending phase. For instance, this is the case of the bit reversal traffic pattern. In a 4-ary 3-tree, the nodes attached to the switch 0, nodes $\langle 0,0,0\rangle$, $\langle 0,0,1\rangle$,$\langle 0,0,2\rangle$ and $\langle 0,0,3\rangle$), send all their messages to

$\langle 0,0,0\rangle$, $\langle 2,0,0\rangle$,$\langle 1,0,0\rangle$,$\langle 3,0,0\rangle$, respectively. As it can be observed, all destinations share the two last components, so packets will share the same link in the first two stages of their ascending path. As it was expected, the performance of the deterministic routing algorithm was poorer than the adaptive one.

5.2.2 I/O Traces

In this section, we analyze the performance of deterministic routing using a more realistic traffic pattern. In particular, we have used the aforementioned I/O traces. We have simulated a 2-ary 7-tree. As the *cello* system, it has a storage subsystem with 23 disks that we attached to 23 leaves of the 2-ary 7-tree. The remaining 105 leaves have been used to attach processing nodes.

The used traces are quite old. Nowadays, I/O traffic has changed. The technology allows faster devices (hosts and storage devices) to be used, and thus heavier traffic could be expected. For this reason, we have applied different time compression factors to the traces.

Figure 7.(a) shows the time required to deliver all the packets included in the trace when there is no need to ensure in-order delivery at different compression factors. As it can be observed, there are no differences between the routing algorithms, and it does not matter the selection function used, because the network is able to deal with the injected traffic. Figure 7.(b) shows the total execution time when in-order delivery is guaranteed using a reorder-buffer size of 8MB. As it can be observed, the differences among the routing algorithms are considerable. When no compression factor is applied, the deterministic routing algorithm reduces execution time by 4.4 over the adaptive one using FF as selection function. If SADP selection function is used, execution time is reduced by a factor of 2.9. If we increase the injection rate by applying a compression factor of 10, we can observe how the improvements obtained by the deterministic routing algorithm over the adaptive one are increased even more. Execution time is decreased by a factor of 30 over FF, and 4 over SADP. In this case, the overhead of in-order delivery does not compensate the flexibility of adaptive routing.

6 Conclusions

Adaptive routing in fat-trees is commonly accomplished by an upwards adaptive phase and a downwards deterministic phase. The deterministic path to follow in the downwards phase depends on the adaptive path followed in the upwards phase. This adaptive routing algorithm, does not guarantee in-order delivery of packets.

In order to guarantee in-order delivery, we have proposed a deterministic routing algorithm for fat-trees that balances traffic very well, both in the upwards and in the downwards phase. It can be easily implemented using the FIR strategy proposed in [7]. When synthetic traffic patterns are used, the deterministic routing algorithm is able to obtain similar performance results to the ones obtained by the adaptive routing algorithm, with the exception of some particular traffic pattern, such as bit reversal. But, when we analyze the performance by using the traffic generated from real applications (I/O traces), we can observe that, when in-order delivery is enforced, the deterministic routing algorithm strongly outperforms the adaptive ones. In particular, the proposed deterministic routing algorithm improves the time to deliver all the messages of the I/O traces over the SADP adaptive proposal by a factor near 3.

Moreover, the deterministic routing algorithm allows a simpler implementation, since neither selection function, nor additional hardware resources to ensure in-order delivery are needed.

References

[1] B. Abali et al. Adaptive routing on the new switch chip for IBM SP systems, *Journal of Parallel and Distributed Computing*, vol. 61, no. 9, pp. 1148-1179, September 2001.

[2] E. Bakker, J. van Leeuwer and R.B. Tan. Linear Interval Routing *Algoritms review*,2. pp. 45-61, 1991.

[3] W.J. Dally and B. Towles. Principles and Practices of Interconnection Networks. *Morgan Kaufmann*, 2004.

[4] J. Duato, S. Yalamanchili and L. Ni. Interconnection Networks. An Engineering Approach. *Morgan Kaufmann*, 2004.

[5] J. Flich, M.P. Malumbres, P. López, J. Duato. Improving Routing Performance in Myrinet Networks. *Proc. 14th International Parallel and Distributed Processing Symp.*, 2000.

[6] F. Gilabert, M.E. Gómez, P. López, J. Duato. On the Influence of the Selection Function on the Performance of Fat-trees. *European Conf. on Parallel Computing*, Aug. 2006.

[7] M.E. Gómez, P. López, and J. Duato. A Memory-Effective Routing Strategy for Regular Interconnection Networks. *IEEE International Parallel and Distributed Processing Symposium*, April 2005. Best Paper Award in the Architecture Track.

[8] Infiniband Trade Association. www.infinibandta.org

[9] J.C. Martinez, J. Flich, A. Robles, P. Lopez, and J. Duato. Supporting Adaptive Routing in IBA Switches. *Journal of Systems Architecture*, 49:441–449, 2004.

[10] J.C. Martinez, J. Flich, A. Robles, P. Lopez, J. Duato and M. Koibuchi. In-Order Packet Delivery in Interconnection Networks using Adaptive Routing. *IEEE International Parallel and Distributed Processing Symp.*, April 2005.

[11] Myricom. www.myri.com

[12] Quadrics homepage. http://www.quadrics.com.

[13] C. Ruemmler, J. Wilkes. Unix Disk Access Patterns. Winter Usenix Conference, Jan 1993.