Towards an Efficient Fat-Tree like Topology

D. Bermúdez Garzón¹, C. Gómez², M.E. Gómez¹, P. López¹, and J. Duato^{1,*}

¹ DISCA Department, Universitat Politècnica de Valencia,
Camino de Vera, 14, 46071-Valencia, Spain
dieberg1@posgrado.upv.es

² Dept. of Computing Systems, University of Castilla-La Mancha, Spain

Abstract. Topology and routing algorithm are two key design parameters for interconnection networks. They highly define the performance achieved by the network, but also its complexity and cost. Many of the commodity interconnects for clusters are based on the fat-tree topology, which allows both a rich interconnection among the nodes of the network and the use of adaptive routing. In this paper, we analyze how the routing algorithm affects the complexity of the switch, and considering this, we also propose and analyze some extensions of the fat-tree topology to take advantage of the available hardware resources. We analyze not only the impact on performance of these extensions but also their influence over switch complexity, analyzing its cost.

Keywords: Regular indirect topologies, fat-trees, adaptive and deterministic routing.

1 Introduction

Cluster machines have become very popular to build high performance computers and data centers in the last years due to their excellent cost-performance ratio. These machines use commodity computers linked by a high-performance interconnection network, which plays a critical role to achieve a high performance. Two of the main design issues of interconnection networks are topology and routing [1]. In deterministic routing schemes, packets traverse a fixed, predetermined path between their source and their destination, while in adaptive routing schemes, packets may use one of the available different alternative paths from their source to their destination. An adaptive routing algorithm is composed of the routing and selection functions [2]. For each packet, the routing function supplies the set of available routing options, while the selection function [3] selects one of them. This selection function usually takes into account the status of the network. Adaptive routing usually helps in balancing network traffic, thus allowing the network to obtain a higher throughput. However, with adaptive routing, in-order packet delivery can not be ensured, which is mandatory for some applications. This is the case, for example, for certain cache coherence protocols, some communication libraries and network technologies. On the other hand, deterministic routing algorithms usually do a very poor job balancing traffic among the network links, due to the

* This work was supported by the Spanish MICINN, Consolider Programme and Plan E funds, as well as European Commission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04.

lack of path diversity, but they are easier to be implemented. Moreover, deterministic routing guarantees in-order packet delivery by design.

Concerning topology, cluster-based machines usually choose either regular direct networks (tori and meshes) or, more frequently, multistage indirect networks (MINs). In particular, fat-trees [4] have raised in popularity in the past few years, since most of the commonly-used interconnect technologies provide support for this topology. Moreover, some of the most powerful machines ever built implement a fat-tree topology, such as, the CM-5, the Cray BlackWidow machine, or the recent number 1 machine in the Top500 list, Tianhe-1A [5].

Routing and topology, besides impacting network performance, also highly define the hardware cost and complexity of the interconnection network. In this paper, we analyze several proposals of topologies and routing algorithms analyzing not only their performance but also estimating the resources needed to implement them, with the aim of taking the highest advantage of the available hardware resources.

2 Motivation

Interconnection networks are often designed only for performance. However, the increase in complexity of interconnection network negatively affects its cost and power consumption. A balanced design should consider both performance and complexity, trying to obtain a good tradeoff between them.

In this paper, we focus on routing in the fat-tree topology and how the routing algorithm can be used to simplify the switch complexity and even modify the topology. Routing in fat-trees has two different subpaths (an upwards subpath and a downwards one). In the commonly-used routing algorithm [6], the upwards path is fully adaptive and the downwards one is deterministic. The unique downwards subpath is determined by the path selected in the upwards one. So, in fat-trees, unlike other topologies, the decisions made in the upwards subpath can be critical, having a strong impact on network performance. In [7], the DESTRO mechanism was proposed to effectively select the upwards path for each packet in a fat-tree, in order to highly reduce the Head-of-Line (HoL) blocking effect [8] and in this way improving performance. This mechanism can be implemented in a very simple way, as only a component of the destination identifier is used for routing, both in the up and down subpaths. In [9], the RUFT topology was proposed which is a simplification of the fat-tree topology taking advantage of the nice properties of DESTRO. RUFT allows to reduce the hardware resources to the half while providing similar performance.

Besides obtaining different levels of performance, fat-trees (with adaptive routing), DESTRO and RUFT require switches with different complexity. Therefore, a fair comparison among them should consider both performance and complexity issues. On the other hand, as the RUFT topology use simpler switches than fat-trees, some extensions with a complexity level similar to the fat-tree can be devised. We propose two extensions. The first one (RUFT-PL) substitutes each original link by two physical parallel links, allowing the use of some adaptivity in the network and at the expense of increasing switch complexity. The second one (RUFT-DB) substitutes each original link by another one with double width, therefore duplicating link bandwidth. The new proposed

extensions of the RUFT topology are able to increase the throughput of the interconnection network by 2.5x compared to the fat-tree while having a similar or even lower cost.

The rest of the paper is organized as follows. Section 3 revisits the fat-tree topology and presents the notation and assumptions used in the following sections. It also describes the adaptive routing algorithm commonly-used in fat-trees and DESTRO analyzing the switch requirements. Section 4 presents RUFT analyzing also its switch requirements. Section 5 presents the RUFT topology extensions proposed in this paper. Section 6 provides several evaluation results for different configurations of topology and routing algorithm. Finally, some conclusions are drawn.

3 Fat-Tree Topology

The fat-tree topology is based on a complete tree that gets thicker near the root. The arity of the switches increases as we go nearer to the root, which makes the physical implementation unfeasible. For this reason, some alternative implementations have been proposed in order to use switches with fixed arity. In particular, the k -ary n -tree [6] is 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 connects 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 $\{0, 1, \dots, k-1\}^n$, 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 $\{0, 1, \dots, k-1\}^{n-1}$ which identifies the switch inside the stage. Two switches $\langle s, o_{n-2}, \dots, o_1, o_0 \rangle$ and $\langle s', o'_{n-2}, \dots, o'_1, o'_0 \rangle$ 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 an edge between the switch $\langle 0, o_{n-2}, \dots, o_1, o_0 \rangle$ and the processing node $p_{n-1}, \dots, p_1, p_0$ if $o_i = p_{i+1}$ for all $i \in \{n-2, \dots, 1, 0\}$. This edge is labeled with p_0 . In what follows, we will assume that descending links are labeled from 0 to $k-1$, and ascending links from k to $2k-1$.

3.1 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 k up output ports. Once a nearest common ancestor has been reached, the packet is turned around and sent downwards to its destination as 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. Once in that stage, 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.

Fig. 1. Ports that can be requested in a 4-ary n -tree using adaptive routing

Switch complexity of fat-trees can be easily computed considering that each switch has k bidirectional input and output ports, leading to $2k \times 2k = 4k^2$ switching elements. However, this rationale does not account for the actual requirements of switching activity of the routing algorithm. As can be observed, in the upwards subpath, at each switch, the k input ports can forward packets through either any of the up k output ports, if the packet continues in its upwards subpath, or any of its down output ports if the packet starts its downwards subpath. On the other hand, in the downwards subpath, there are k input ports that can only request k down output ports, since once a packet has started its downwards subpath, the packet must continue going downwards. Figures 1a and 1b show the output ports that can be requested in the upwards and downwards directions, respectively, in the switches of a 4-ary n -tree.

A common way of implementing switches is by using as many multiplexers as the number of required output ports. Each multiplexer has a number of inputs equal to the number of input ports that can request the corresponding output port. In the switch we are considering, ports in the upwards direction require k multiplexers with k inputs each one or a $k \times k = k^2$ complexity. On the other hand, ports in the downwards direction require k multiplexers with $2k$ inputs each one or a $k \times 2k = 2k^2$ complexity. Total switch complexity can be easily obtained as the sum of the upwards and downwards directions complexities, leading to a the switch complexity of $3k^2$ switching elements.

3.2 DESTRO Routing in Fat-Trees

Contrary to the previously presented routing algorithm, DESTRO [7] is deterministic, that is, in both subpaths there is only one path for each source–destination pair. We selected this algorithm due to its good results and also because it is able to highly reduce the switch complexity. The high performance is due to the appropriate selection of packet upwards subpaths which distribute destinations in a very effective way to highly reduce the HoL blocking effect. The packet downwards subpaths are determined by the upwards subpath followed by the packet and with DESTRO the interferences among different destinations in the packet downwards subpaths are completely eliminated. All the packets destined to a particular node are kept inside the same sub-tree (See Figure 2a), and have a unique and exclusive down path. This is performed by using the destination identifier to select one of the multiple available upwards subpaths. In DESTRO, the output port for routing a packet in a particular switch is given both by the destination identifier and the stage where the switch is located. In particular, it considers the component of the packet destination corresponding to that stage (i.e., a switch located at

Fig. 2. Deterministic routing in a 2-ary 3-tree

stage s considers the s^{th} component of the destination identifier, that is p_s). Therefore, at the switch $\langle s, o_{n-2}, \dots, o_1, o_0 \rangle$, the selected output port for a packet with destination $p_{n-1}, \dots, p_1, p_0$ will be $k + p_s$.

Figure 2a shows the destination node distribution in the ascending and descending links of a 2-ary 3-tree using DESTRO. In the first stage, the least significant component of the packet destination identifier (the least significant bit in this example) is used to select the ascending output port. At the second stage, the destinations of all packets that reach a given 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 next one in the destination address. For instance switch 4 is only reached by packets destined to nodes 0, 2, 4 and 6. These nodes have the same least significant component, which is 0. Of them, only packets destined to nodes 4 ($\langle 100 \rangle$) and 6 ($\langle 110 \rangle$) must be forwarded upwards. Packets destined to node 4 will select the first up link, and packets destined to node 6 the another one. Following this mechanism in all the upwards stages, finally, packets destined to a particular destination reach the same switch at the last stage and have a unique down subpath. Figure 2a highlights all the paths to node 7 and how all of them share the same downwards subpath.

By using DESTRO, the switch complexity can be highly reduced. The upwards switch activity is the same as in the fat-tree with adaptive routing. That is, each input port in the upwards subpath can request either any of the up output port, or any of the down output ports. However, in the down subpath, each link, input port and output port is used exclusively by packets sent to a unique destination. As a consequence, a given input port will always request the same output port, since all the packets that arrive to a particular input port, in their downwards subpath, are destined to the same node and are always forwarded to the same output port. This allows a noticeable reduction in switch complexity. Using multiplexers to implement switches, ports in the upwards direction require k multiplexers, each with k inputs, or a $k \times k = k^2$ complexity, and ports in the downwards direction require k multiplexers with $k + 1$ inputs (the k upwards ports plus the unique downwards one) or $k \times (k + 1) = k^2 + k$ switching elements. Therefore the required switch complexity required by DESTRO is $2k^2 + k$.

4 RUFT

The RUFT topology [9] is a simplification of the fat-tree topology obtained by taking advantage of the nice properties of DESTRO. In particular, since there is no switching activity in the downwards subpath, the switches are simplified by making them unidirectional. Therefore, the whole downwards subpath is transformed in links that connect the last stage to the different processing nodes; (see Figure 2b). Notice also that, as the topology is unidirectional, there is not chance to start the downwards subpath before reaching the last stage and, therefore, the paths are longer since all the packets must reach the last stage, contrary to the fat-tree topology where depending on the source–destination pair, different number of stages must be traversed.

Using RUFT, the switch complexity corresponds to the switch complexity of a unidirectional switch of k input ports and k output ports, where any of the k input ports can request any of the k output ports, so the switch complexity is k^2 . As it can be seen, the switch complexity has been reduced more than twice when comparing it with DESTRO and three times when comparing with the fat-tree with adaptive routing.

5 Extensions of the RUFT Topology

In this section, taking into account the difference in switch complexity among RUFT, DESTRO, and fat-trees, we propose two different enhancements of the RUFT topology that, by using more complex switches, improve its performance. The idea is to obtain a new topology with a switch complexity similar to DESTRO and fat-trees, which will allow us to perform a fair comparison among all the feasible choices.

Both of them, as RUFT, are unidirectional MINs. Therefore, all packets will traverse the same number of hops (the number of network stages) regardless of their source–destination pair. The first proposal (which will be referred to as RUFT-Parallel Links or RUFT-PL) uses switches with a number of ports equal to the number of ports of the fat-tree switches. As fat-tree ports are bidirectional, RUFT-PL have twice number of ports than RUFT. Therefore, each pair of switches are connected by two parallel links. On the other hand, the number of network links is also equal to the number of network links of the fat-tree topology. The second proposal (which will be referred to as RUFT-Double Bandwidth or RUFT-DB) uses switches with the same number of ports as the RUFT topology, but these ports and their associated links have double width (thus double link bandwidth) than the ones of RUFT and fat-tree topologies. The idea is to make the network bandwidth equal to the one of the fat-tree, while maintaining the number of ports equal to the one of the RUFT switches.

5.1 RUFT with Parallel Links

RUFT-PL uses the same number of switches as the fat-tree topology, as the RUFT topology does, but they have the same number of ports as the fat-tree switches. Our proposal pursues to implement the RUFT topology but without reducing the switch complexity. As fat-tree switches have bidirectional ports, RUFT-PL switches can double the number of ports of RUFT switches, leading to a complexity of $2k \times 2k$ switching elements.

We propose to use the available k additional ports to have two parallel links connecting each pair of switches of the original RUFT topology. These two parallel links provides additional routing flexibility that can be exploited in different ways.

A first approach is to distribute different destinations among them and, in this way, the remaining HoL blocking that still appears in the upwards subpaths of the RUFT topology is reduced even more. To do so, we propose that, at a given switch, the pair of output channels (i.e. output ports) to be used will be given by the destination component corresponding to the stage of the switch (as RUFT and DESTRO do) but the parallel channel of the port that will be finally used will be given by the least significant bit of the next destination component. Notice that we use the next destination component because it will be used in the next stage to select the output port (the pair of output ports in RUFT-PL). Therefore, packets that will take different output ports in the next switch will be forwarded through different channels, thus reducing interferences among them.

Formally, at switch $\langle s, o_{n-2}, \dots, o_1, o_0 \rangle$, the selected pair of output ports for a packet with destination node $p_{n-1}, \dots, p_1, p_0$ will be $k + p_s$, as in RUFT, and from this output port, the parallel channel to be used will be given by $p_{s+1} \bmod 2$ to select the least significant bit of the next destination node component. Figure 3 shows how the destination nodes are distributed among the output links of switches in RUFT-PL, for $k = 2$ and $n = 3$. Notice that the least significant bit of the next destination node component is used to decide the parallel link to be used by a given destination at a particular set of output ports. As can be seen, thanks to classifying destination nodes in this way, each input port of a given switch only requests a particular set of output ports. This is better shown in Figure 4a, where the output ports requested by the two parallel channels of input port 0 are shown. Figure 3 suggests that the last stage is not required since packets with different destinations already arrive through different channels. However, this is only true for $k = 2$, if $k > 2$ is used, the last stage is required. In Figure 4b another example for $k = 4$ is shown. In this case, a given input port only requests the two channels of two different output ports.

In RUFT-PL, each input port can only request the k output channels associated to $k/2$ output ports, whose implementation require k multiplexers with $2k$ inputs, which leads to a complexity of $k \times 2k = 2k^2$, which is even smaller than the one required by DESTRO. We will refer to this way of using parallel links as RUFT-PL-C, given that packets are classified in different channels of the parallel link according to its destination. The way RUFT-PL-C selects parallel output channels may not be the best option under some non-uniform traffic patterns. For these traffic patterns, in most of the switches only one of the two parallel channels will be actually used. For this reason, we have also evaluated another version of RUFT-PL that selects the link of the parallel channel following other criteria, without classifying packets. We will refer to this approach as RUFT-PL-NC. One possible criterion can be selecting the channel with more free buffer. The key point of RUFT-PL-NC is that it can always use both parallel channels per port regardless what kind of traffic is used. However, in this case, the switch complexity is increased since each input channel can ask for any of the up output ports, so the switch complexity is $2k \times 2k = 4k^2$ switching elements, which is higher than the complexity of the fat-tree switch with adaptive routing. Table 1 summarizes the switch complexities of the different proposals.

Fig. 3. A 2-ary 3-tree RUFT-PL

Table 1. Switch complexity comparison

Routing	Switch complexity
Adaptive	$3k^2$
DESTRO	$2k^2 + k$
RUFT	k^2
RUFT DB	$2k^2$
RUFT PL C	$2k^2$
RUFT PL NC	$4k^2$

5.2 RUFT with Double Bandwidth

RUFT-DB is a slight modification made on the RUFT topology, where we keep the same number of ports per switch, but doubles the bandwidth of all links connecting the switches by making them wider. On the other hand, the size of the buffers of the input and output are also doubled. The idea is to have the same bandwidth as the one of the fat-tree topology and also the same buffer resources. That is, RUFT-DB is a RUFT topology where buffers and link bandwidth are doubled. Regarding switch complexity, the number of required switching elements are the same as in RUFT, but as each one has twice the width of RUFT, the required switch complexity measured in RUFT switching elements is $2k^2$.

6 Evaluation

6.1 Simulation Environment

In order to compare the performance of the different topologies and routing algorithms analyzed in this paper, we have developed an event-driven simulator. The simulator models a virtual cut-through network composed by a set of switches. Each switch has a routing control unit that applies the routing algorithm and configures its internal crossbar. Links allow pipelined transmission and are characterized by the delay required by a flit to reach the edge of the link (fly time) and the interval among consecutive flit transmissions (link delay). Routing, crossbar, fly and link delays has been assumed equal to one clock cycle. In the case of RUFT, RUFT-DB and RUFT-PL, an increased fly time has been considered in the long links that connect the last stage to the processing nodes. This time is modelled as the product of one fly time by the number of network stages. Three packet sizes have been considered (8B, 64B and 4KB). Several synthetic traffic were considered: uniform, bit-reversal and hotspot with a traffic concentration of 5% and 15%.

6.2 Performance Results

In Figure 5a we compare the different analyzed configurations with uniform traffic and 64B packets in 2-ary 4-tree topologies. As can be seen, fat-tree with adaptive routing

(referred to as FTA), DESTRO and RUFT obtain similar throughputs, being DESTRO the best of them and RUFT the worst one since its switches have half the number of ports than the fat-trees switches (FTA and DESTRO). It also has half number of links (unidirectional links are used) than FTA and DESTRO. On the other hand, concerning the three RUFT extensions proposed in this paper, both RUFT-PL configurations and RUFT-DB are able to double the throughput obtained by the first three configurations. In particular, RUFT-PL-NC is the one that achieves the highest throughput, whereas the other two configurations obtain similar throughputs, having RUFT-DB a lower network latency except near the saturation point. The effect of a greater link bandwidth of RUFT-DB compensates the partial adaptivity provided by RUFT-PL. Only at very high loads, the additional routing freedom is worth. Concerning the two versions of RUFT-PL, classifying packets between the two parallel links does not seem a good idea. The explanation is that although by classifying packets HoL effect is further reduced, it is also true that the number of routing options is limited to only one versus two in the non-classify option. Indeed, traffic in RUFT is already classified by design, and the additional distribution offered by splitting destinations among the two links of the parallel channel does not compensate the reduction in routing flexibility, as only one of the parallel links can be used for a given destination. Finally, notice though, that the RUFT extensions require more complex switches than RUFT. An additional comment can be stated regarding the fact that configurations that allow multiple paths among source-destination pairs (i.e., those ones that provide adaptive routing), which include not only FTA but also RUFT-PL-NC, may introduce out-of-order delivery of packets.

Influence of Network Size. In this section, we analyze the influence of network size on the performance of the different analyzed configurations. Uniform traffic and 64B packets were considered. Figures 5a to 5e show results for k -ary n -trees with different number of stages and arities, increasing accordingly the network size. As expected, as the number of stages increases, the latency of the different configurations also increases, since more stages must be crossed by packets. Indeed, the performance is slightly worse for the configurations with higher arity ($k=4$) because larger switches mix more destinations in each link, thus contributing to increase the HoL blocking effect. Anyway, the relative behavior of the different analyzed configurations remain the same. Notice that RUFT-PL-C gets slightly worse as the number of stages increases. Again, the less flexibility for routing is amplified by the fact of traversing more stages (i.e., more routings are performed to reach a given destination). The configuration that achieves the highest throughput is RUFT-PL-NC, and RUFT-DB is the one that obtains the lowest latency before saturation due to the fact that data is transmitted twice faster through links and switches.

Impact of Packet Size. In this section, we analyze how the behavior of the different analyzed configurations are affected by the packet size. We have tested several network sizes with different packet sizes, figure 6a and 6b shows the results for a 2-ary 6-tree with a packet size of 8 bytes and 4 Kbytes. As expected, latency increases with the packet size and so does network throughput as the contribution of the transmission time of packets is by far more important than the one that depends on the number of hops. The relative positions of the different configurations are the same regardless

Fig. 4. Output ports that can be requested by an input port in RUFT switches

Fig. 5. Network latency versus throughput for different network sizes with uniform traffic and a packet size of 64B

of the packet size, with some advantage of RUFT-DB with large packets, taking full advantage of the increased link bandwidth. Anyway, the best configuration is always RUFT-PL-NC.

Effect of Traffic Pattern. Figure 7 shows the results for 2-ary 5-tree topologies with a packet size of 64 bytes with other traffic patterns. For the bit-reversal traffic pattern, RUFT and RUFT-PL-C obtains the worst results. In this case, the paths provided by RUFT collide among different source-destination pairs and the additional channels that the PL extension supplies are useless in most cases as packets are always sent to the same node and the channel of the parallel link are selected according to the destination node. FTA is able to improve performance thanks to the use of alternative paths. The additional routing flexibility of RUFT-PL-NC also helps in improving network throughput. Finally, once more, the doubled channel bandwidth allows RUFT-DB to double the network throughput reached by RUFT.

(a) 2-ary 6-tree, Packet Size 8B.

(b) 2-ary 6-tree, Packet Size 4KB.

Fig. 6. Network latency versus throughput for two network sizes and different packet sizes with uniform traffic

(a) Bit-Reversal.

(b) HotSpot 5%.

(c) HotSpot 15%.

Fig. 7. Network latency versus throughput with different traffic patterns in 2-ary 5-tree networks and a packet size of 64B

When considering hotspot traffic (see Figures 7b and 7c), we can observe that the higher the severity of the hot-spot, the lower the achieved network throughput. Notice the unusual shape of the plots. Once the paths that reach the hot-spot are saturated, only traffic destined to other nodes move through the network, until the network is completely saturated. The final saturation point reached is the same obtained for the uniform traffic pattern (see Figure 5b). The results also confirm RUFT-PL-NC and RUFT-DB as the best routing algorithms.

Effect of Routing and Fly Times. In this section we analyze how the obtained conclusions may change if the path setup time was higher than we considered. Path setup time is given by the sum of routing, switch and fly times. We have analyzed the impact of a higher routing time, which should impact all configurations and fly time, which should especially impact the long RUFT links of the last stage.

Figure 8a shows the results of a increased routing time in a 2-ary 5-tree with uniform traffic. As it can be seen, the higher average distance of fat-trees leads to a higher latency values of FTA and DESTRO. Moreover, the throughput of all the configurations is strongly reduced, as packets spend most of its time at the routing control units. Figure 8b shows the effect of a high fly time. As expected, all the RUFT configurations are strongly penalized, increasing their average latency. On the other hand, RUFT-DB becomes the best configuration for the full range of traffic. When traversing a link takes a long time, transmitting more information per time unit allows amortizing this time.

Fig. 8. Network latency versus throughput with uniform traffic in 2-ary 5-tree networks and a packet size of 64B

Performance/Cost Evaluation. This section analyzes and compares the hardware cost of each configuration evaluated in the paper. In particular, we consider for each configuration, the number of links, the number of switches and the number of switching elements required. In particular, the number of switching elements is the most appropriate measure to compare the different configurations since it considers both, the number of switches and its degree. Moreover, we define two different figures of merit that take into account both, performance and cost of a given configuration. The first one is the throughput per switching element. The higher the value of this parameter, the better performance/cost ratio has the configuration. To account for the latency, we also obtain the product of the latency by the number of switching elements. Although this value could be obtained for every value of traffic analyzed, for the sake of shortness, only results for very low load traffic are shown. In this case, the higher the value, the worse the configuration.

A subset of the results are shown in Table 2. As it can be seen, the cheapest topology is RUFT, as it has the lowest number of links and switching elements. RUFT uses half the number of links and switching elements of the RUFT-PL-C configurations and even less compared to the others. Nevertheless, RUFT does not reach the best throughput. RUFT-PL-NC is able to double the throughput obtained by FTA, DESTRO and RUFT, and outperforms the other two RUFT extensions proposed in this paper by 6%. If we consider both performance and cost, as shown by the defined figures of merit, results confirm that the best configurations are RUFT-PL-NC and RUFT-DB, in this order,

Table 2. Performance-Cost on different network sizes with several algorithms and uniform traffic

	Topology	Links	Switching Elements	Throughput	Throughput/ Switch Element	Low Load Latency	Base Lat./ Switch Element
2-ary 5-tree	FTA	320	960	0,6401	0,000667	94,4	90643
	DESTRO	320	800	0,6746	0,000843	95,4	76313
	RUFT	192	320	0,6103	0,001907	91,7	29349
	RUFT-DB	384	640	1,3611	0,002127	52,9	33835
	RUFT-PL-C	384	640	1,2892	0,002014	90,0	57623
	RUFT-PL-NC	384	1280	1,4107	0,001102	89,0	113950
4-ary 4-tree	FTA	2048	12288	0,5076	0,000041	90,6	1113818
	DESTRO	2048	9216	0,5250	0,000057	92,3	850411
	RUFT	1280	4096	0,4712	0,000115	87,2	357389
	RUFT-DB	2560	8192	1,0618	0,000130	48,6	398076
	RUFT-PL-C	2560	8192	1,0033	0,000122	85,2	697777
	RUFT-PL-NC	2560	16384	1,1993	0,000073	84,0	1376813

closely followed by RUFT. RUFT-PL-NC and RUFT-DB are able to increase by 3x the performance obtained by FTA, and by nearly 2.5x the DESTRO one. On the other hand, when the latency is considered, the one that obtains the lowest performance–cost is by far RUFT–DB, due to the fact that packet are sent faster.

7 Conclusions

In this paper, we analyze and compare several topologies and routing algorithms for fat-tree-related (fat-tree and RUFT) topologies. As RUFT topology requires less network resources than fat-trees, in order to perform a more fair comparison, we propose some extensions to the RUFT topology that tries to match the resources committed in the fat-tree. The final goal is to obtain a good tradeoff between performance and required resources. In particular, RUFT–DB uses links that are twice as wide as RUFT ones, therefore increasing network bandwidth, also doubling the required number of switching elements. The RUFT–PL–C and RUFT–PL–NC proposals also enhances the network by aggregating two links connecting switches, therefore increasing routing flexibility but at the cost of a higher number of switching elements. RUFT–PL–C tries to reduce the number of resources needed by restricting routing, statically classifying destinations among the aggregated links, also reducing the HoL blocking effect.

We evaluated all the proposals for different network sizes and network loads, obtaining that the RUFT–PL–NC approach, which adds routing flexibility to RUFT, is the one that obtains the best network throughput while the RUFT–DB one, which adds channel bandwidth helps in reducing packet latency. Most important, when we combine cost (measured in number of network resources) and performance, results show that RUFT–DB, RUFT–PL–NC and RUFT are very cost–effective configurations, obtaining the best throughput per switching element values. When latency–number of switching elements is consider, RUFT–DB is by far the best option due to its increased channel bandwidth.

References

1. Dally, W., Towles, B.: Principles and Practices of Interconnection Networks. Morgan Kaufmann (2003)
2. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks. An Engineering Approach. Morgan Kaufmann (2004)
3. Gilabert, F., Gómez, M.E., López, P., Duato, J.: On the influence of the selection function on the performance of fat-trees, pp. 864–873 (2006)
4. Leiserson, C.E.: Fat-trees: Universal networks for hardware-efficient supercomputing. In: ICPP, pp. 393–402 (1985)
5. Tianhe-1a, <http://www.nscg-tj.gov.cn/en/>
6. Petrini, F., Vanneschi, M.: k-ary n-trees: High performance networks for massively parallel architecture. IEEE Micro 15 (1995)
7. Gómez, C., Gilabert, F., Gómez, M.E., López, P., Duato, J.: Deterministic versus adaptive routing in fat-trees. In: 21th Int. Parallel and Distributed Processing Symposium (April 2007)
8. Nachiondo, T., Flich, J., Duato, J.: Buffer management strategies to reduce hol blocking. IEEE Trans. Parallel Distrib. Syst. 21(6), 739–753 (2010)
9. Gómez, C., Gilabert, F., Gómez, M.E., López, P., Duato, J.: Ruft: Simplifying the fat-tree topology (December 2008)