Applying In-Transit Buffers to Boost the Performance of Networks with Source Routing

José Flich, *Member*, *IEEE*, Pedro López, *Member*, *IEEE Computer Society*, Manuel Perez Malumbres, *Member*, *IEEE*, José Duato, *Member*, *IEEE*, and Tomas Rokicki

Abstract—Clusters of workstations (COWs) are becoming increasingly popular as a cost-effective alternative to parallel computers. In these systems, processors are connected using irregular topologies, providing the wiring flexibility, scalability, and incremental expansion capability required in this environment. Myrinet is one of the most popular interconnection networks for COWs. Myrinet uses source routing and wormhole switching. The up*/down* routing algorithm is used to build the network routes. On the other hand, in Myrinet, network behavior is controlled by the software running at the network interfaces. Hence, new features such as new routing algorithms can be added by only changing this software. In previous work, we proposed the In-Transit Buffer (ITB) mechanism to improve the performance of source routing-based networks. The ITB mechanism temporarily ejects packets from the network at some intermediate hosts and later reinjects them into the network, performing a special kind of virtual cut-through switching at these hosts. We applied this mechanism to up*/down* routing, in order to remove the down $\rightarrow$ up forbidden channel dependences that prevented minimal routing between every pair of hosts. Results showed that network throughput can be more than doubled on medium-sized (32 switches) networks. In this paper, we analyze in depth the effect of using ITBs in the network, showing that they not only serve for guaranteeing minimal routing, but also that they are a powerful mechanism able to balance network traffic and reduce network contention. To demonstrate these capabilities, we apply the ITB mechanism to improved routing schemes, such as DFS and smart-routing. These routing algorithms (without ITBs) are able to improve the performance of up*/down* by 30 percent and 90 percent, respectively, for a 32-switch network. The evaluation results show that, when ITBs are used together with these improved routing algorithms, network throughput achieved by DFS and smart-routing can still be improved by 56 percent and 23 percent, respectively. However, smart-routing requires a time to compute the routing tables that rapidly grows with network size, it being impossible in practice to build networks with more than 32 switches. This high time is mainly motivated by the need of obtaining deadlock-free routing tables. However, when ITBs are used, one can decouple the stages of computing routing tables and breaking cycles. Moreover, as stated above, ITBs can be used to reduce network contention. In this way, in this paper, we also propose a completely new routing algorithm that tries to balance network traffic by using a simple and low time consuming strategy. The proposed algorithm guarantees deadlock freedom and reduces network contention with the use of ITBs. The evaluation results show that our algorithm obtains unprecedented throughputs in 32-switch networks, tripling the original up*/down* and almost doubling smart-routing.

Index Terms—Networks of workstations, irregular topologies, wormhole switching, minimal routing, source routing.

1 Introduction

Due to the increasing computing power of microprocessors and the high cost of parallel computers, Clusters Of Workstations (COWs) are currently being considered as a cost-effective alternative for small-scale parallel computing. Although COWs do not provide the computing power available in multicomputers and multiprocessors, they meet the needs of a great variety of parallel computing problems at a lower cost. The interconnection network used is usually the local area network (LAN) all computers are attached to. Research in interconnection networks for COWs is advancing relatively rapidly due to the research effort made on interconnects for parallel computers.

The evolution of COWs is closely related to that of local area networks (LANs). LANs have migrated from shared medium to point-to-point links. As an example, consider the evolution of the Ethernet family up to recent Gigabit Ethernet networks [20]. Although Ethernet is very popular, other commercial LANs have arisen in the high-speed networking arena, trying to provide solutions for some of the Ethernet weaknesses such as quality of service, priority-based traffic, gigabit channels, and flow control mechanisms (ATM, VG100AnyLan, Autonet, Myrinet).

In COWs, topology is usually fixed by the physical location constraints of the computers, making the resulting topology irregular. On the other hand, either source or distributed routing may be used to send packets across the network. In source routing, the path to destination is built at the source host and it is written into the packet header before transmission. Switches route packets through the fixed path found at the packet header. One example of network with source routing is Myrinet [1]. In distributed routing, the packet header only includes the destination host. So, to forward a packet, each switch decides the next output channel that will be used. The Autonet network [18] is an example of a network with distributed routing. In networks with source routing, switches are simpler and faster than those in distributed routing because no route

- *J. Flich, P. López, M.P. Malumbres, and J. Duato are with the Department of Computer Engineering, Universidad Politécnica de Valencia, Camino de Vera, 14, 46071-Valencia, Spain.*
  E-mail: {jflich, plopez, mperez, jduato}@gap.upv.es.
- *T. Rokicki is with Instantis, 725 B Loma Verde, Palo Alto, CA 94303.*
  E-mail: rokicki@instantis.com.

Manuscript received 14 Aug. 2001; accepted 5 June 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114777.

decisions have to be made in them. However, with distributed routing, the path followed by a packet can be dynamically changed in order to avoid congested areas, thus taking advantage of adaptivity.

An important design issue related to routing is deadlock handling. Usually, avoidance techniques based on a deadlock-free routing algorithm are used. Hence, routing is restricted so that there are no cycles in the channel dependence graph (CDG) [5]. A less restrictive approach that also fits into this category allows the existence of cyclic dependences between channels while providing some escape paths to avoid deadlock [7], [22]. Many routing algorithms that follow these approaches have been proposed.

Up*/down* [18] is one of the best known routing algorithms for irregular networks. It was used in the DEC AN1 system [15] and currently on Myrinet networks [1]. Up*/down* routing is quite simple. It is based on an assignment of direction labels to links. To do so, a breadth-first spanning tree is computed and, then, the *up* end of each link is defined as: 1) The end whose switch is closer to the root in the spanning tree; 2) the end whose switch has the lower ID, if both ends are at switches at the same tree level. As a result, each cycle in the network has at least one link in the *up* direction and one link in the *down* direction. Cyclic dependences between channels are avoided by prohibiting packets from traversing links in the *up* direction after having traversed one in the *down* direction. While up*/down* routing is simple, it concentrates traffic near the root switch and uses a large number of nonminimal paths.

Adaptive-trail routing [16] is based on finding a Eulerian trail in the network, which establishes a dependence order for all channels. Shortcut channels are added to the original Eulerian trail in order to reduce path lengths. The routing rule allows the use of shortcut channels if they are free; otherwise, the Eulerian trail channels are used as escape channels to avoid deadlock. Adaptive-trail routing was proposed to be used in networks with distributed routing.

Smart-routing [3] first computes all the possible paths for every source-destination pair, also building the CDG. Then, it searches through the CDG for cycles. An iterative process breaks cycles by removing dependences, taking into account a heuristic cost function. This process finishes when the CDG has no cycles. Although smart-routing distributes traffic better than other approaches, it has the drawback of its high computation overhead since it uses a linear programming solver to balance the traffic while it tries to break cycles.

Minimal adaptive routing [21] multiplexes each link into two virtual channels, the *adaptive* and *escape* channels, respectively. The *adaptive* channel is used to route packets through minimal paths without restrictions, while the *escape* channel is used as an escape path when the *adaptive* channel is busy. As proposed in [21], *escape* channels must be used following the up*/down* rule, which guarantees an acyclic CDG. Minimal adaptive routing is only suitable for networks with distributed routing. This routing algorithm outperforms the up*/down* routing algorithm by providing minimal paths in most cases. However, since it cannot be applied to networks with source routing and it requires the use of virtual channels, minimal adaptive routing is not well suited for current LANs.

Finally, the DFS routing algorithm [17] computes a depth-first spanning tree. Then, it adds the remaining channels to provide minimal paths, which may lead to cycles in the CDG. As in previous approaches, cycles are broken by restricting routing. However, channels are labeled using a heuristic in such a way that the number of routing restrictions is small.

Routing algorithms that lead to an acyclic CDG can be used both in networks with source routing and in networks with distributed routing. Up*/down* routing, smart-routing, and DFS routing belong to this category. On the other hand, routing algorithms that rely on escape paths to avoid deadlocks can only be used in networks with distributed routing because the source host cannot know in advance if an escape channel will be needed at some intermediate switches in the path to the destination. This is the case of adaptive-trail and minimal adaptive routing. In this paper, we will only focus on networks with source routing schemes.

2 Background and Motivation

All previous source-based routing strategies (up*/down*, DFS, and smart-routing) use different algorithms to compute the possible paths among source-destination pairs while maintaining deadlock freedom (acyclic CDG). Although cycles are removed, routing is restricted and many of the allowed paths are not minimal, increasing both latency and contention in the network. Also, in routings based on spanning trees (up*/down* and DFS), forbidding some paths may result in unbalanced network traffic, which will lead to rapid saturation. Moreover, as network size increases, routing algorithms tend to increase the number of routing restrictions, leading to the use of even longer paths, higher network contention, and more unbalanced traffic. On the other hand, the smart-routing algorithm highly balances traffic. However, this good traffic balance also leads to the use of long paths and, most importantly, it has a high computation cost.

Additionally to these problems, another limiting factor of performance in such networks is network contention. Because wormhole switching is used and virtual channels are not allowed, contention on one link can rapidly block other links, cascading throughout the network. This effect increases latency and reduces overall performance.

Taking into account all of these problems in networks with source routing, the best routing algorithm should be the one that: 1) always uses short paths, 2) highly balances network traffic, and 3) helps in reducing network contention while guaranteeing deadlock freedom. Finally, this algorithm should be scalable and require a reasonable computation time.

In previous work [8], we proposed a new mechanism (in-transit buffers, ITB) for networks with source routing. Basically, this mechanism avoids routing restrictions by ejecting packets at intermediate hosts and later reinjecting them. This mechanism can be easily implemented in Myrinet by modifying the network control program at the network interface card without changing the network hardware.

This mechanism was originally proposed to provide minimal routing to up*/down*. In this routing algorithm, ITBs are put in all the *down-up* transitions. The mechanism has been extensively evaluated for both irregular [8] and regular networks [9] under different traffic patterns, network topologies, network sizes, and different message sizes. Overall, this mechanism improves on the performance achieved by up*/down*. Moreover, as network size increases, more benefits are obtained since the up*/down* routing does not scale well.

DFS and smart-routing obtain better performance than up*/down* routing. As the ITB mechanism can be applied to any source-based routing algorithm, it will be interesting to analyze if it can also boost the performance of these routing algorithms even more. As a first objective, in this paper, we will apply the in-transit buffer mechanism to DFS and smart-routing.

From [8], [9], we concluded that, by avoiding routing restrictions, the ITB mechanism allows the use of minimal paths for every source-destination pair. Moreover, it also allows a good traffic balance not achieved by up*/down*.

On the other hand, besides applying the ITB mechanism to traditional routing algorithms, this mechanism can also be used to build new routing algorithms. As this mechanism avoids routing restrictions, we can compute any set of paths and, by using ITBs to break cycles, we can obtain a set of deadlock-free paths (with ITBs) ready to be used in any network with source routing. Therefore, as a second objective of the paper we will propose a new routing algorithm for networks with source routing that will try to optimize the use of in-transit buffers.

Traditionally, existing routing algorithms have always been focused on providing short paths and a good traffic balance. However, to the best of our knowledge, network contention and computation time have never been addressed when designing a routing algorithm. Our new routing algorithm will consider the following:

- **Minimal paths**. By always using minimal paths, messages use the minimum number of resources in the network, thus lowering network contention. Therefore, our new routing algorithm will be restricted to always using minimal paths for every source-destination pair.
- **Network contention**. As mentioned above, the ITB mechanism affects network contention by temporarily ejecting messages from the network. Moreover, network contention is affected by the number of ITBs we put in the network. More ITBs in the network will reduce network contention even more. On the other hand, the main drawback of the mechanism is the latency added to packets that use ITBs. With more ITBs in the network, packets will use, on average, more ITBs and latency will increase. However, this latency penalty is only noticeable for short packets and low traffic conditions [9]. Therefore, there is a trade off between the number of ITBs and latency. We will study the effect on network contention of using a different number of ITBs in the network, observing the increase in latency. We will propose different methods to compute and allocate ITBs in the network.
- **Traffic balance and computation time**. Routing algorithms that have load balancing in mind, like smart-routing, can have a high computation cost. Without the advantages of ITBs or another deadlock prevention mechanism, these algorithms must yield a set of paths free of deadlocks. Deadlock avoidance means that these algorithms must often deal with many nonminimal paths, increasing the time to compute balanced paths significantly. For example, the smart-routing algorithm is unable to obtain paths for networks with more than 32 switches in a reasonable amount of computation time. However, by using ITBs to break cycles, path computation can be performed without taking deadlocks into account. Therefore, all the possible paths can be evaluated at the same time in order to select the set of paths that offer a good traffic balance. So, by using ITBs we can design a traffic balancing algorithm that, while perhaps suboptimal, offers good load balancing, is simple, and can be quickly computed.

To sum up, the objective of the paper is twofold: First, ITBs will be applied to optimized routing algorithms (DFS and smart-routing) to analyze if network performance can be increased and, second, a new routing algorithm will be presented. This algorithm will use minimal paths, will obtain a good traffic balance, and will lower network contention. This algorithm will rely on ITBs.

The rest of the paper is organized as follows: Section 3 presents how the ITB mechanism works and gives some details about its implementation. In Section 4, the mechanism is applied to some optimized routing strategies. In Section 5, we present the new routing algorithm. In Section 6, evaluation results for different networks and traffic load conditions are presented, first analyzing the benefits of using our mechanism combined with previous routing proposals and second analyzing the benefits of our new routing algorithm. Finally, in Section 7, some conclusions are drawn.

3 ITBs: A Mechanism to Remove Channel Dependences

The basic idea of the mechanism is to break cyclic dependences with host buffering. The paths between source-destination pairs are computed following any given rule and the corresponding CDG is obtained. Then, the cycles in the CDG are broken by splitting some paths into subpaths. To do so, an intermediate host inside the path is selected and used as an in-transit buffer (ITB); at this host, packets are ejected from the network as if it were their destination. The mechanism works similarly to the cut-through switching technique. Therefore, packets are re-injected into the network as soon as possible to reach their final destination. Notice that the dependences between the input and output channels of the switch are completely removed because, in the case of network contention, packets will be completely ejected from the network at the intermediate host. The CDG is made acyclic by repeating

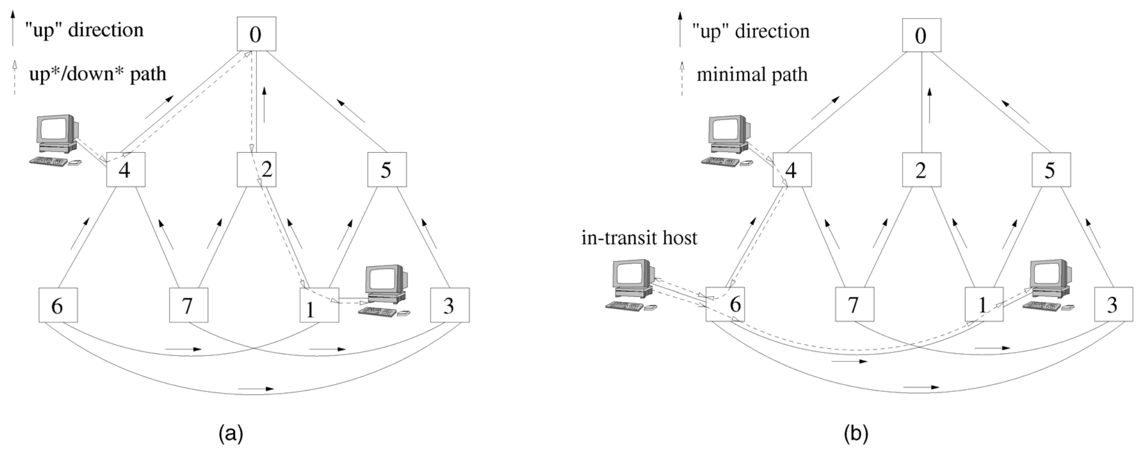


Fig. 1. Link direction assignment and use of the ITB mechanism for an irregular network.

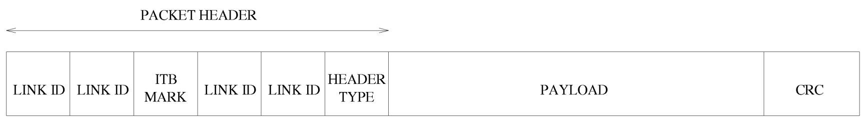


Fig. 2. Myrinet packet header for routing schemes based on ITBs.

this process until no cycles are found. Notice that more than one intermediate host may be needed for a particular path.

This mechanism was first proposed for the up*/down* routing algorithm to remove the dependences between *down* and *up* channels [8]. It was referred to as *in-transit buffers* (ITB). By removing dependences between *down* and *up* channels, minimal paths between some hosts, forbidden by the original up*/down* routing algorithm, are now provided. As an example, Fig. 1a shows a network and the assignment of link directions following the up*/down* rule. Although there is a minimal path between switch 4 and switch 1 ($4 \rightarrow 6 \rightarrow 1$), it is forbidden because it uses an *up* link after a *down* link at switch 6. However, with the ITB mechanism (see Fig. 1b), this path is allowed by using one host at switch 6 as an in-transit host to break the dependence. By using ITBs, minimal routing can be guaranteed while keeping deadlock freedom.

On the other hand, ejecting and reinjecting packets at some hosts also improves performance by reducing network contention. Packets that are ejected free the channels they have reserved, thus allowing other packets requiring these channels to advance through the network (otherwise, they would block). As a consequence, network throughput may be increased.

Hence, the goal of the ITB mechanism is not only to provide minimal paths by breaking some dependences, but also to improve performance by reducing network contention. However, ejecting and reinjecting packets at some intermediate hosts also introduces some penalty to them. First, the latency of these packets is increased. When the network load is low, this effect will be more prominent. An efficient implementation of the mechanism can help to keep this overhead low. Second, ITBs use some additional resources in both network (links) and network interface cards (memory pools and DMA engines). Finally, in order to use the mechanism, there must be hosts attached to each switch when there is a routing restriction. Although this is a common situation where hosts are spread over an irregular network, in the case where there are no hosts attached to a critical switch, the routing algorithm can still use nonminimal valid up*/down* paths.

If the rules used to build the paths between source-destination pairs lead to an unbalanced traffic distribution, then adding more ITBs than the ones strictly needed will also help. This is the case for up*/down* because this routing algorithm saturates the area near the root switch. Thus, there is a trade off between using the minimum number of ITBs that guarantees deadlock-free minimal routing and using more than these to improve network throughput through better traffic balance.

Notice that this mechanism does not require virtual channels. However, if virtual channel flow-control is used, the mechanism can also be used. By using virtual channel flow-control, network contention is lower. However the same nonminimal path set will be used and, more importantly, the same traffic unbalance will be obtained. Therefore, the use of ITBs in networks with virtual channel flow-control will also help to obtain shorter paths and a better traffic balance. In this paper, we focus only on networks without virtual channels.

Let us discuss the implementation of the ITB mechanism on a Myrinet network. In order to implement the ITB mechanism, changes in the packet header format and in the Myrinet Control Program (MCP) [11] are required. In Fig. 2, we can see the new header format that supports in-transit buffers. The entire path to destination is built at the source host. A mark (ITB mark) is inserted in order to notify the in-transit host that the packet must be reinjected into the network after removing that mark. After the mark, the path from the in-transit host to the final destination (or to another in-transit host) follows.

Fig. 3 shows the implementation of the in-transit buffer mechanism. First, some memory is needed at the network interface card to store in-transit packets and the MCP

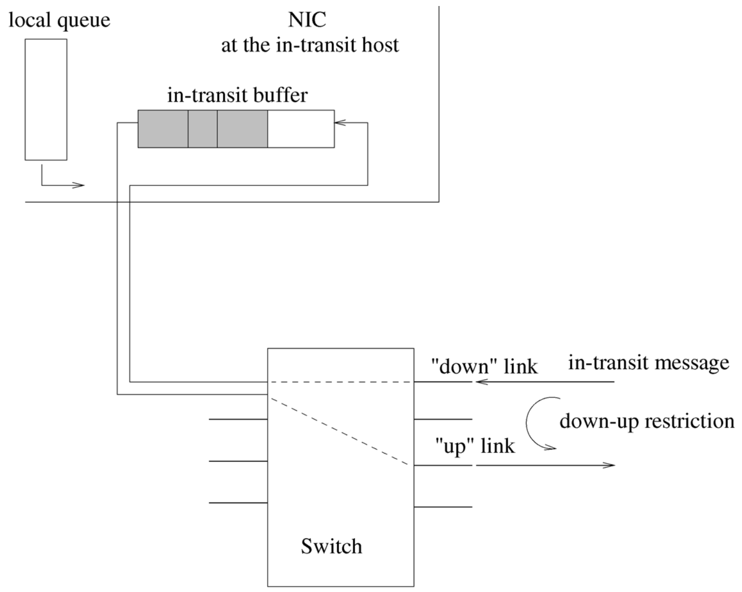


Fig. 3. In-transit buffer mechanism.

program has to be modified to detect in-transit packets and process them accordingly. In order to minimize the introduced overhead, as soon as the in-transit packet header is processed and the output channel is free, a DMA transfer can be programmed to reinject the in-transit packet. Thus, the delay to forward this packet will be independent of packet length and will be equal to the time required for processing the header and starting the DMA (when the output channel is free). As the MCP allows this kind of DMA programming, it is possible to efficiently implement the in-transit buffer mechanism without modifying the Myrinet network hardware. On the other hand, there is no problem if the DMA transfer begins before the packet has been completely received because it will arrive at the same rate that it is transmitted,[1] assuming that all the links in the network have the same bandwidth.[2] Also, an easy detection mechanism can be implemented in order to detect, at the destination host, possible overruns due to DMA transfers.

Notice that Myrinet does not implement virtual channels. Therefore, once a packet header reaches the network interface card, data will continue arriving at a constant rate. The only additional requirement is that the packet is completely stored in the network adapter memory at the source host before starting transmission to avoid interference with the source host I/O bus.

To make this mechanism deadlock-free, it must be guaranteed that an in-transit packet that is being reinjected can be completely ejected from the network if the reinjected part of the packet becomes blocked, thus removing potential channel dependences ($down \rightarrow up$ transitions) that may result in a deadlock configuration. So, when an in-transit packet arrives at a given host, care must be taken to ensure that there is enough buffer space to store it at the interface card before starting its reinjection. If the buffer space at the network interface card has been exhausted, the MCP should either store the packet in the host memory or discard it. In practice, a very small number of buffers are required. In fact, in all the simulations run (see Section 6), the reserved memory space (512KB) was enough to handle all the in-transit packets without using the host memory.

In order to quantify the memory requirements of the ITB mechanism, in [10], we obtained the number of in-transit packets that are simultaneously stored at each NIC for different packet injection rates. In particular, we analyzed an irregular 32-switch network using a uniform traffic pattern with 32-byte packets. Simulation was run for a period of time long enough to deliver 2 million of packets. Results showed that, even when the network is approaching saturation, the amount of memory needed at the NICs was negligible. No more than 15 packets (480 bytes without considering headers) were held at the same time at a particular NIC. When the network was saturated, more space was required to store in-transit packets. However, the maximum number of ITB packets at a particular NIC for a 2-million packet simulation was lower than 3,000 (96KB without considering headers). The same behavior was obtained for other network topologies and packet sizes.

Taking into account that current NICs offer up to 8MB of available memory, less than 128KB are set aside for the MCP, and that network load will not be usually beyond saturation, we can conclude that the probability of NIC memory overflow is very low. So, in the case of NIC memory overflow, the best solution will be to discard the packet.

The use of the mechanism on a particular in-transit host could affect the performance of its local traffic. The in-transit host output channel will be shared among local and in-transit packets. An excessive number of in-transit packets could block the injection of local packets. In order to minimize this effect, a dynamic priority scheme has been implemented. With this mechanism, local traffic priority is

1. Due to limited memory bandwidth in the network interfaces, a source host may inject *bubbles* into the network, thus lowering the effective reception rate at the in-transit host. This problem has been addressed and can be easily avoided when implementing the MCP code. Also, future implementations of Myrinet interfaces will eliminate this problem.

2. Myrinet supports mixing links with different bandwidth.

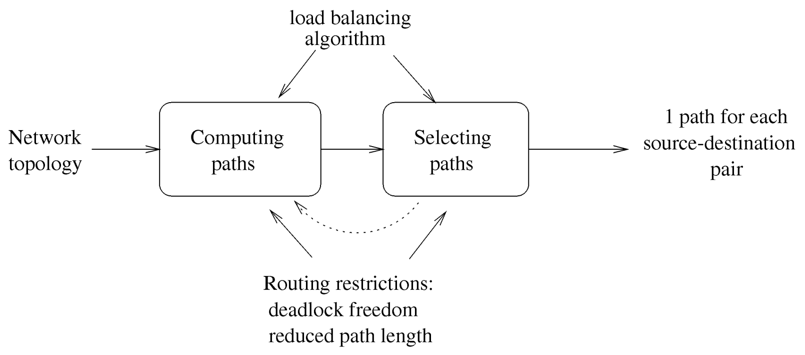


Fig. 4. Stages in the design of typical source routing algorithms.

increased as the number of queued local packets increases. In particular, when this number is large, both kinds of packets have the same priority. By using this strategy, in all the simulations run, starvation was avoided in all nodes.

Some preliminary work has been done to implement the mechanism on a Myrinet network [4]. This version has been developed by modifying the GM message layer protocol [11]. The implementation adds the possibility of sending packets through ITB hosts while allowing all the benefits from the GM software. The correctness of the mechanism has been verified in order to work properly under different workload configurations.

4 Applying the ITB Mechanism to Any Routing Algorithm

Although the ITB mechanism was first proposed for the up*/down* routing, it can be applied to any source-based routing scheme. In this section, we will apply the ITB mechanism to several source-based routing algorithms: up*/down*, DFS, and smart-routing. In the case of up*/down*, we will use the two approaches mentioned before: allocating the minimum and nonminimum number of ITBs that guarantee deadlock-free minimal routing. In the first case, given a source-destination pair, we will compute all the minimal paths. If there is a valid minimal up*/down* path, it will be chosen. Otherwise, a minimal path with ITBs will be used. In the second approach, we will use more ITBs than strictly needed to guarantee deadlock-free minimal routing. In particular, we will randomly choose one minimal path. If the selected path complies with the up*/down* rule, it is used without modification. Otherwise, ITBs are inserted. Notice that there may exist valid minimal up*/down* paths between the same source-destination pair, but they are not used. Hence, more ITBs than the ones strictly necessary will be used. By using these two approaches, we can evaluate the trade off mentioned above. In the case of DFS, we will use ITBs in the same way as in the second approach used for up*/down*, but verifying if the paths comply with the DFS rule.

However, for smart-routing, we will use a different approach. We first compute the paths between source-destination pairs that better balance network traffic (in the same way as smart-routing does). It is important to notice that the obtained routes are not the same as those that smart-routing computes. Smart-routing computes both balanced and deadlock-free routes, whereas we compute only balanced routes. For this reason, we will refer to these routes as "balanced" rather than "smart." Then, we compute the CDG and place ITBs where needed to convert it into an acyclic one. Since computing balanced routes alone is easier than computing both balanced and deadlock-free routes, the computational cost of the resulting routing algorithm is lower than the computational cost of smart-routing. Hence, the resulting routing algorithm relaxes one of the most claimed drawbacks of smart-routing.

5 A New Routing Algorithm with ITBs

In order to ease the description of the new routing algorithm, we first decompose traditional routing algorithms for networks with source routing into a few well-defined stages, as shown in Fig. 4. First, routing algorithms compute some possible paths for every source-destination pair. After computing the different paths, the final set of paths that will be used is selected. Usually, only one path for every source-destination pair is selected. These two stages (computing and selecting paths) are influenced by some restrictions. The first one is deadlock freedom. This restriction can be applied to the first stage when routes are computed (up*/down* and DFS). Therefore, only paths that do not introduce cycles in the CDG are computed. However, this restriction can be also applied to the second stage (smart-routing). The smart-routing algorithm considers all the possible paths for every source-destination pair and then selects the set of paths that does not introduce cycles in the CDG.

The second restriction imposed at these stages is that short path lengths are preferred. However, by keeping the deadlock freedom condition, short paths are sometimes not allowed and, therefore, this restriction is not usually accomplished and, then, the routing algorithm must choose longer paths.

Finally, paths are computed taking into account some kind of traffic balance. So, a load balancing algorithm is used in order to select paths. The deadlock freedom condition usually restricts the number of alternative paths that can be computed and, therefore, traffic balance may not be achieved (up*/down* and DFS). On the other hand, to override this problem, some routing algorithms (smart-routing) have some feedback between the two stages. Therefore, selecting some paths can involve computing

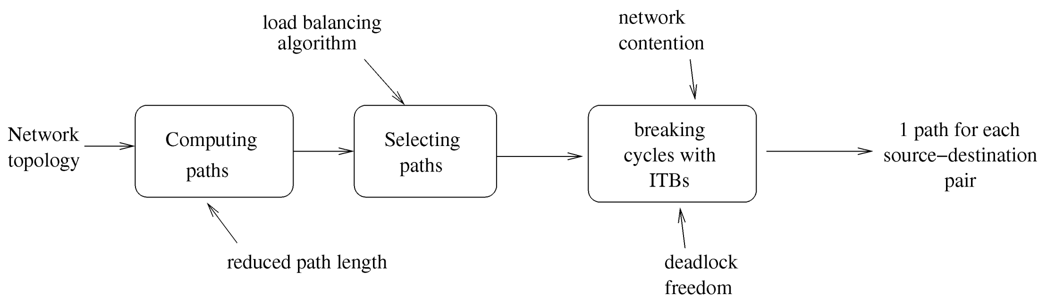


Fig. 5. Stages in the new routing algorithm for networks with source routing.

and changing some already computed paths. This feedback can introduce a high computation time penalty. On the other hand, traffic balance is usually achieved because more alternative paths are considered.

By using ITBs, we can change the chain design shown in Fig. 4. In particular, our new routing algorithm will follow the stages shown in Fig. 5.

First, the routing algorithm computes the whole set of minimal paths. Notice that, in this first stage, there is no restriction in keeping the deadlock freedom condition (this will be ensured in the last stage). In the next stage, traffic is balanced. As deadlock related conditions are not considered, we can play with more paths and, therefore, we can try to balance traffic without requiring feedback between stages.

After selecting the final paths, the routing algorithm goes to the last stage, where in-transit buffers are inserted in order to break cycles. Notice that, by using more ITBs, we can also reduce network contention. Finally, we have just one minimal path for every source-destination pair and the resulting path set is balanced and does not introduce cycles in the CDG thanks to the use of ITBs.

Our new routing algorithm will try to optimize each stage. In the computing paths stage, it will consider all the possible minimal paths for every source-destination pair unless there are a large number of such paths, in which case we will randomly select a reasonably large number of paths. The path selection stage will balance paths using a new traffic balancing algorithm. Once the paths are computed and selected, the routing algorithm will apply ITBs in order to break cycles and, at the same time, it will reduce network contention. Different algorithms for breaking cycles will be presented.

In the next two sections, we explain the algorithms used for balancing paths and lowering network contention, respectively.

5.1 Balancing and Selecting Paths

Usually, routing algorithms balance traffic having in mind a uniform distribution of message destinations. Although actual traffic patterns change depending on running applications, this message destination distribution is often considered as a worst case (from the point of view of locality). In fact, the smart-routing algorithm balances traffic for a uniform distribution of message destinations and performs well for different traffic patterns (refer to Section 6). We will take the same approach in this paper, balancing paths assuming a uniform distribution of message destinations.

The way we balance traffic is by counting the number of paths that cross each link and trying to keep that number uniform on all network links. Therefore, once paths are computed, we select those paths that best balance the number of paths per link.

Fig. 6 shows the algorithm used for balancing and selecting paths. This algorithm removes paths minimizing a cost function. The cost function is the standard deviation of the number of paths that cross each link. To do that, the algorithm removes a path at each loop iteration. In order to decide which path will be removed, the algorithm removes one path temporarily and computes the standard deviation of the number of paths per link using the rest of paths. This is repeated for all the paths. After this process, the algorithm knows the standard deviations resulting for the elimination of each path. Then, it removes the path that shows the maximum decrease (minimum increase) in the standard deviation. Notice that a given path will be removed only if there are still alternative paths for that source-destination pair. The algorithm finishes when none of the path can be removed. At the end, there will be only one path for each source-destination pair.

This simple algorithm may not achieve the optimal traffic balance. However, it should balance traffic better than those used in the up*/down* and DFS routing algorithms due to the lack of routing restrictions imposed by the requirements of deadlock freedom.

5.2 Breaking Cycles and Lowering Network Contention

The last stage of our routing algorithm deals with cycles in the CDG and also with network contention. As explained before, there is a trade off between network contention and latency penalty when adding ITBs.

The best solution for reduced network contention is to put ITBs on all the switches for all the paths. So, each packet will be ejected and reinjected at each switch. This approach obtains an acyclic CDG because there are only dependences between a channel connecting two switches and a channel connecting a switch to a host and vice versa. This approach will reduce network contention to the minimum.

On the other hand, the best solution for reducing latency overhead is to put the minimum number of ITBs that guarantees deadlock freedom. One alternative is to search cycles in the CDG, breaking them by inserting ITBs. This algorithm does not guarantee a minimum number of ITBs since the number of cycles found depends on the order in

```
procedure compute-paths()
var
    p:path
begin

    repeat
        p==nil
        for each path
            eliminate path from weight-map
            compute standard deviation of weight-map
            add path to weight-map
            annotate the path that shows the lowest standard deviation and has alternative paths (p)
        end

        if p<>nil
            eliminate path from weight-map
            delete path (p)
        end
    until p==nil

end procedure.
```

Fig. 6. Algorithm for balancing and selecting paths.

which they are broken. However, the algorithm will use a low number of ITBs.

In order to fully explore the trade off between reduction in network contention and latency increase, we will also evaluate three approaches that fall between the latter two approaches by using different quantities of ITBs. These approaches will put, on average, 33 percent, 50 percent, and 66 percent of the maximum number of ITBs in the network. All of these approaches must be also designed taking into account that the CDG must be acyclic.

To use 33 percent of ITBs while breaking all cycles, we use the following rules: First, we assign a random identifier to each switch, taking care to assign unique identifiers. Then, for each subpath that crosses three switches in sequential order (*A*, *B*, and *C* with identifiers *a*, *b*, *c*), we put an ITB if

$$(b > a) \quad AND \quad (b > c). \tag{1}$$

Taking into account that identifiers are generated randomly and are unique, we can obtain six orderings of switches (*abc, bac, bca, cba, cab, acb*). Two (*cab,acb*) out of these six orderings follow the rule (1). Hence, there is a 33 percent of probability of placing an ITB.

With these rules, cycles are removed since, in each cycle, there will be at least one switch with higher identifier than the neighbor switches of the cycle.

For 66 percent ITBs, we follow similar rules. By using random and unique identifiers, we put an ITB whenever the identifier of switch B is higher than either identifier of switch A or identifier of switch C. That is,

$$(b > a) \quad OR \quad (b > c). \tag{2}$$

Orderings *abc*, *cba*, *cab*, and *acb* follow the rule (2). Therefore, probability of putting an ITB is 66 percent. Also, cycles are prevented since, in each cycle, there will be at least one switch with an identifier higher than one of its neighbors.

Finally, for 50 percent ITBs, we put an ITB whenever a path uses a switch with a lower identifier than the following one. That is,

$$a < b. \tag{3}$$

All of these approaches will be evaluated in the next section with the balancing algorithm presented above.

6 Performance Evaluation

In this section, we evaluate the behavior of the ITB mechanism over the different source-based routing algorithms described above. We also evaluate the new routing algorithm. For this algorithm, we study the effect on traffic balancing and also the effect on network contention.

First, we present the network model, and the network loads used throughout the evaluation. Then, we present some considerations about the methodology used to evaluate the mechanism. Finally, we present the simulation results.

6.1 Network Model

The network is composed of a set of switches and hosts, all of them interconnected by links. Network topologies are completely irregular and have been generated randomly, taking into account only three restrictions. First, we assume that there are exactly four hosts connected to each switch. Second, all the switches in the network have the same size. We assume that each switch has eight ports. So, there are four ports available to connect to other switches. Finally, two neighboring switches are connected by a single link. These assumptions are quite realistic and have already been considered in other evaluation studies [21], [22].

In order to evaluate the influence of network size on system performance, we vary the number of switches in the network. We use network sizes of 16, 32, and 64 switches. Thus, there are 64, 128, and 256 hosts in the system, respectively. To make results independent of the topology, we evaluate up to 10 random topologies for each network size.

Links, switches, and interface cards are modeled based on the Myrinet network. Concerning links, we assume Myrinet short LAN cables [14] to interconnect switches and workstations. These cables are 10 meters long, offer a bandwidth of 160 MB/s, and have a delay of 4.92 ns/m (1.5 ns/ft). Flits are one byte wide. Physical links are also one flit wide. Transmission of data across channels is pipelined [19]. Hence, a new flit can be injected into the physical channel every 6.25 ns and there will be a maximum of eight flits on the link at a given time.

We do not use virtual channels since the actual Myrinet switches do not support them. A hardware "stop and go" flow control protocol [1] is used to prevent packet loss. In this protocol, the receiving switch transmits a stop(go) control flit when its input buffer fills over (empties below) 56 bytes (40 bytes) of its capacity. The slack buffer size in Myrinet is fixed at 80 bytes.

Each Myrinet switch has a simple routing control unit that removes the first flit of the header and uses it to select the output link. That link is reserved when it becomes free. Assuming that the requested output link is free, the first flit latency is 150 ns through the switch. After that, the switch is able to transfer flits at the link rate, that is, one flit every 6.25 ns. Each output port can process only one packet header at a time. An output port is assigned to waiting packets in a demand-slotted round-robin fashion. When a packet gets the routing control unit, but it cannot be routed because the requested output link is busy, it must wait in the input buffer until its next turn. A crossbar inside the switch allows multiple packets to traverse it simultaneously without interference.

Each Myrinet network interface card has a routing table with one or more entries for every possible destination of messages. The way tables are filled determines the routing scheme. For each source-destination pair, only one path will be computed for the different routing algorithms.

In the case of using in-transit buffers, the incoming packet must be recognized as in-transit and the transmission DMA must be reprogrammed. We have used a delay of 275 ns (44 bytes received) to detect an in-transit packet and 200 ns (32 additional bytes received) to program the DMA to reinject the packet.[3] Also, the total capacity of the in-transit buffers has been set to 512KB at each interface card.

6.2 Network Load

In order to evaluate different workloads, we use different message destination distributions to generate network traffic:

- Uniform distribution. The destination of a message is chosen randomly with the same probability for all the hosts. This pattern has been widely used in other evaluation studies [2], [6].
- Bit-reversal distribution. The destination of a message is computed by reversing the bits of the source host identification number. This pattern has been selected taking into account the permutations that are usually performed in parallel numerical algorithms [12], [13].
- Local distribution. Message destinations are, at most, five switches away from the source host and are randomly computed.
- Hot-spot distribution. A percentage of traffic is sent to one host. The selected host is chosen randomly. The same host number will be used for all the topologies. In order to use a representative hot-spot distribution, we have used different percentages of the traffic sent to the hot-spot host depending on the network size. In particular, we have used 20 percent, 15 percent, and 5 percent for 16, 32, and 64-switch networks, respectively. The rest of the traffic is randomly generated using a uniform distribution.
- Combined distribution. This distribution mixes the uniform, bit-reversal, local, and hot-spot distributions. Each host will generate message destinations using each distribution with the same probability. With this distribution, we model the traffic generated by different applications that are concurrently running in the system.

Although we use different message sizes (32, 512, and 1K bytes), for the sake of brevity, results will be shown only for 512-byte messages.

6.3 Methodology

Although the ITB mechanism can be implemented in software in Myrinet, simulation was preferred rather than a real implementation in order to analyze in depth the behavior of the mechanism in a full range of network configurations.

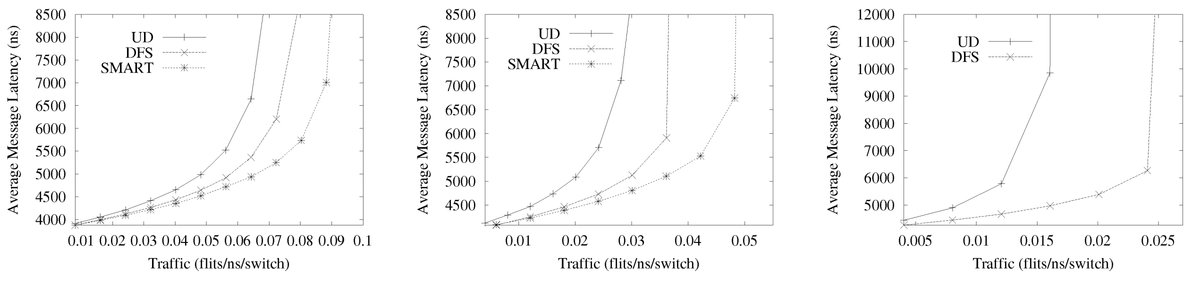
Execution-driven simulations have been avoided since the best benefit of our mechanism is the increase in network throughput. The traffic conditions needed to reach such throughput are very difficult to obtain in execution-driven simulations. Also, network size is restricted in these kinds of simulations due to the prohibitive execution time and to the huge amount of memory needed at the simulation platform. Therefore, execution-driven simulations are not well suited to testing our mechanism in large system configurations.

We use event-driven simulation to evaluate the mechanism. For each simulation run, we assume that the packet generation rate is constant and the same for all the hosts. Once the network has reached a steady state, the flit generation rate is equal to the flit reception rate. We evaluate the full range of traffic, from low load to saturation. During the first 100,000 messages, the system is considered to be unstable. After that, 100,000 more messages are simulated and taken into account in order to obtain results.

6.4 Simulation Results: ITBs over Routing Algorithms

In this section, we evaluate the performance of the in-transit buffer mechanism when it is applied to the source-based routing algorithms mentioned in Section 4. First, we analyze the behavior of the routing algorithms without using in-transit buffers. We evaluate up*/down* routing (UD),

3. These timings have been measured on a real Myrinet network. Average timings have been computed from the transmission of more than 1,000 messages using the Real Time Clock register (RTC) of the Myrinet interface card.



(a) (b) (c)

Fig. 7. Average message latency versus accepted traffic. Message size is 512 bytes. Uniform distribution. Network size is (a) 16 switches, (b) 32 switches, and (c) 64 switches.

depth-first search spanning tree-based routing (DFS), and the smart-routing algorithm (SMART).

Then, we evaluate the use of in-transit buffers over up*/down* and DFS routing. For up*/down* routing, we analyze the two approaches mentioned above: using the minimum number of ITBs needed to guarantee deadlock-free minimal routing (UD_MITB) and using more ITBs (UD_ITB). For DFS routing, we only use the second approach. This new routing algorithm will be referred to as DFS_ITB. Finally, we evaluate the use of in-transit buffers over balanced but deadlocking routes supplied by the smart-routing algorithm. This routing will be referred to as BALANCED_ITB.

For all the evaluations, we use three random topologies of sizes 16, 32, and 64 switches, respectively. We will plot the average message latency[4] measured in nanoseconds versus the accepted traffic[5] measured in flits/ns/switch.

In order to make results independent of the topology, we average the results for 10 random topologies for each network size. We show the increase in throughput[6] when using the in-transit buffer mechanism with respect to the original routing algorithms. Moreover, we show the minimum, maximum, and average increase in throughput.

6.4.1 Routing Algorithms without ITBs

Fig. 7 shows the behavior of the different routing algorithms (UD, DFS, and SMART) for a uniform distribution of message destinations for different topologies of 16, 32, and 64 switches, respectively. SMART routing is not shown for the 64-switch network due to its high computation time.

As was expected, the best routing algorithm is SMART. It achieves the highest network throughput for all the topologies we could evaluate. In particular, for the 16-switch network, SMART increases throughput over UD and DFS routings by factors of 1.22 and 1.10, respectively. Also, for larger networks (32 switches), SMART increases network throughput by factors of 1.77 and 1.28 with respect to UD and DFS routing, respectively. For the 64-switch network, the best routing algorithm is DFS (SMART was not available). Concerning only DFS and UD routing algorithms, we observe that DFS improves over UD for all network sizes.

The higher network throughput achieved by SMART routing is due to its better traffic balancing. Fig. 8 shows the utilization of links connecting switches for the 32-switch network when using UD, DFS, and SMART routing, respectively. Links are sorted by utilization. Traffic is 0.03 flits/ns/switch. For this traffic value, UD routing is reaching saturation. We observe that, when using the UD routing, half the links are poorly used (52 percent of links with a link utilization lower than 10 percent) and a few links highly used (only 11 percent of links with a link utilization higher than 30 percent), some of them being overused (three links with a link utilization higher than 50 percent). Traffic is clearly unbalanced among all the links. On the other hand, DFS routing reduces this unbalancing and has 31 percent of links with link utilization lower than 10 percent and 9 percent of links with link utilization higher than 30 percent. However, the best traffic balancing is achieved by SMART routing. We observe that, for the same traffic value, links are highly balanced, link utilization ranging from 7.76 percent to 20.26 percent (76 percent of links with a link utilization between 15 percent and 20 percent). As traffic is better balanced, more traffic can be handled by the SMART routing and, therefore, higher throughput is achieved.

Table 1 shows minimum, maximum, and average increases of network throughput when comparing UD, DFS, and SMART routing algorithms using 10 random topologies for each network size. We observe that SMART routing always increases network throughput with respect to UD and DFS routing. For 32-switch networks, on average, SMART routing improves over UD and DFS by factors of 1.59 and 1.26, respectively. We also observe that, as a network grows, DFS routing also increases its improvement over UD. For large networks (64-switch networks), DFS improves UD by a factor of 1.38 on average.

Table 2 shows factors of throughput increase for different network traffic patterns and different network sizes. We observe that, for all the traffic patterns analyzed, SMART routing improves, on average, over UD and DFS. For the combined traffic pattern on 32-switch networks, SMART routing improves, on average, over UD and DFS by factors of 1.50 and 1.28, respectively. On the other hand, DFS routing always achieves, on average, higher network

4. Latency is the elapsed time from the injection of a message into the network at the source host until it is delivered at the destination host.

5. Accepted traffic is the amount of information delivered by the network per time unit. In order to make it independent of the number of switches in the network, it is measured in flits/ns/switch.

6. Throughput is the maximum amount of information delivered by the network per time unit. It is equal to the maximum accepted traffic.

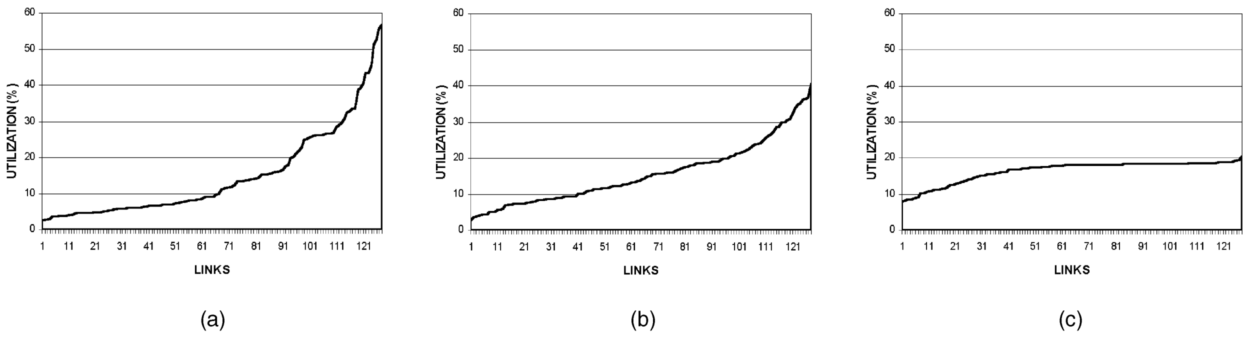


Fig. 8. Link utilization. Traffic is 0.03 flits/ns/switch. Network size is 32 switches. Message size is 512 bytes. Uniform distribution. (a) UD, (b) DFS, and (c) SMART routing.

throughput than UD. In particular, when the combined traffic pattern is used, DFS improves performance over UD on average by a factor of 1.35 for 64-switch networks.

We conclude that traffic balancing plays a key role in obtaining high network performance. SMART routing is the best routing algorithm because it highly balances traffic among all the network links. Its main drawback is the high computation time required to compute the routing tables. The other routing algorithms achieve less throughput, but, on the other hand, they can be quickly computed for any network size. If this is considered a serious constraint, the DFS routing algorithm should be preferred.

6.4.2 In-Transit Buffers on UD and DFS

We focus now on the performance of the in-transit buffer mechanism when it is applied to UD and DFS routings. Fig. 9 shows the performance results obtained by the resulting routing algorithms (UD_MITB, UD_ITB, and DFS_ITB) for the uniform distribution of message destinations for 16, 32, and 64-switch networks.

As can be seen, the in-transit buffer mechanism always improves network throughput over both original routing algorithms. Moreover, as network size increases, more benefits are obtained by the in-transit buffer mechanism. Concerning UD_MITB routing, we observe that UD is improved by factors of 1.12, 1.50, and 2.00 for 16, 32, and 64-switch networks, respectively. However, when more ITBs are used, more benefits are obtained. In particular, UD_ITB improves over UD by factors of 1.22, 2.14, and 2.75 for the 16, 32, and 64-switch networks, respectively. Concerning DFS, DFS_ITB routing improves network throughput over DFS by factors of 1.10, 1.39, and 1.54 for the same network sizes.

When comparing routing algorithms that use more in-transit buffers (UD_ITB and DFS_ITB), we observe that network throughput is roughly the same. These routing algorithms use the same minimal paths and the main difference between them is where the in-transit buffers are allocated and how many in-transit buffers are needed. We also notice that DFS_ITB routing exhibits lower average latency than UD_ITB. This is because DFS routing is less restrictive than UD routing and, therefore, DFS_ITB needs fewer ITBs on average than UD_ITB. When using DFS_ITB routing in the 64-switch network, messages use 0.3 ITBs on average, while the average number of ITBs per message is 0.55 in UD_ITB. This also explains the higher network throughput achieved by UD_ITB since more messages using ITBs are removed from the network, thus reducing network contention.

UD and DFS are computed from a spanning tree. One of the main drawbacks of such an approach is that, as network size increases, a smaller percentage of minimal paths can be used. For the 16-switch network, 89 percent of the paths computed by UD are minimal. However, for 32 and 64-switch networks, the percentage of minimal paths goes down to 71 percent and 61 percent, respectively. When DFS routing is used, something similar occurs. There are 94 percent, 81 percent, and 70 percent of minimal paths for the 16, 32, and 64-switch networks, respectively. When using in-transit buffers, all the computed paths are minimal. Therefore, as network size increases, network throughput increases with respect to routing algorithms that do not use ITBs.

However, the most important drawback of routing algorithms computed from spanning trees is the unbalanced traffic. As network size increases, routing algorithms tend to overuse some links (links near the root switch) and this leads to an unbalanced traffic situation. As in-transit buffers allow the use of alternative paths, network traffic is not forced to pass through the root switch (in the spanning tree), thus achieving better network performance. Fig. 10a, Fig. 10b, and Fig. 10c show the link utilization for UD_MITB, UD_ITB, and DFS_ITB routing, respectively, in the 32-switch network. Network traffic is 0.03 flits/ns/switch (where UD routing saturates). We observe that UD_MITB routing achieves better traffic balancing than UD (refer to Fig. 8a). Only 33 percent of links have a link utilization lower than 10 percent and only 10 percent of links are used more than 30 percent of the time. However, as this algorithm uses ITBs only when needed to ensure deadlock-free minimal routing, a high percentage of paths are still valid minimal up*/down*

TABLE 1
Factor of Throughput Increase between UD, DFS, and SMART Routing for the Uniform Distribution

	SMART vs UD			SMART vs DFS			DFS vs UD		
Sw	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
16	1.11	1.72	1.36	1.10	1.22	1.16	0.99	1.43	1.17
32	1.33	1.92	1.59	1.12	1.36	1.26	1.05	1.50	1.26
64	N/A	N/A	N/A	N/A	N/A	N/A	1.19	1.62	1.38

Message size is 512 bytes.

TABLE 2
Factor of Throughput Increase between UD, DFS, and SMART Routing for Different Traffic Patterns

		SMART vs UD			SMART vs DFS			DFS vs UD		
Distribution	Sw	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Hot-spot	16	0.98	1.19	1.10	0.86	1.18	1.07	0.87	1.17	1.03
Hot-spot	32	1.00	1.40	1.21	0.98	1.17	1.02	0.86	1.42	1.15
Hot-spot	64	N/A	N/A	N/A	N/A	N/A	N/A	1.09	1.70	1.34
Bit-reversal	16	1.02	2.21	1.46	0.87	1.57	1.18	0.97	1.99	1.26
Bit-reversal	32	1.12	2.01	1.69	1.12	1.61	1.30	1.00	1.66	1.39
Bit-reversal	64	N/A	N/A	N/A	N/A	N/A	N/A	1.20	1.75	1.55
Local	16	1.00	1.54	1.27	1.01	1.51	1.21	0.82	1.36	1.06
Local	32	1.17	1.41	1.34	1.12	1.43	1.24	0.93	1.24	1.09
Local	64	N/A	N/A	N/A	N/A	N/A	N/A	0.93	1.04	1.00
Combined	16	1.09	1.75	1.34	1.08	1.30	1.19	0.91	1.37	1.13
Combined	32	1.21	1.73	1.50	1.18	1.37	1.28	0.90	1.29	1.18
Combined	64	N/A	N/A	N/A	N/A	N/A	N/A	1.06	1.57	1.35

Message size is 512 bytes.

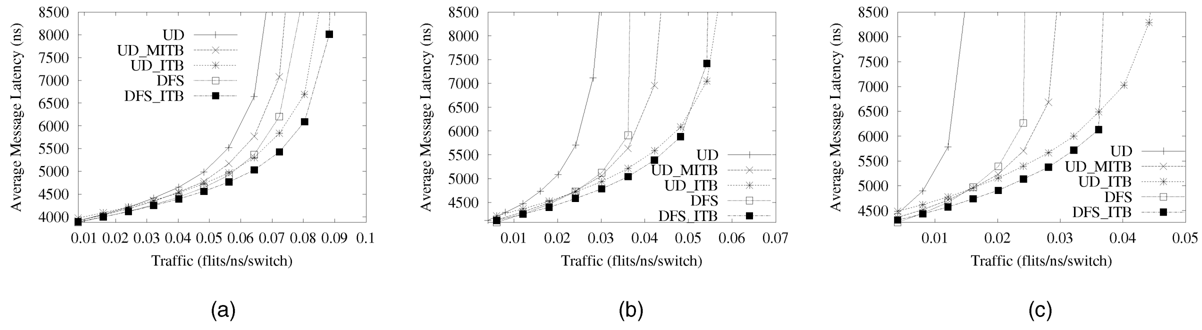


Fig. 9. Average message latency versus accepted traffic. UD, DFS, UD_MITB, UD_ITB, and DFS_ITB routing. Message size is 512 bytes. Uniform distribution. Network size is (a) 16 switches, (b) 32 switches, and (c) 64 switches.

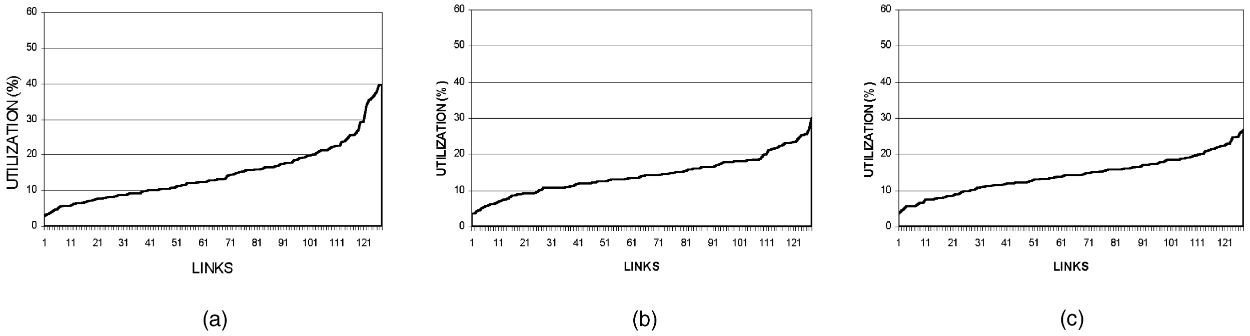


Fig. 10. Link utilization. Network size is 32 switches. Message size is 512 bytes. Traffic is 0.03 flits/ns/switch. Uniform distribution. (a) UD_MITB, (b) UD_ITB, and (c) DFS_ITB.

paths and, therefore, part of the traffic is still forced to cross the root switch. UD_MITB traffic balancing is improved by UD_ITB and DFS_ITB. UD_ITB routing has all the links with a utilization lower than 30 percent and only 20 percent of links are used less than 10 percent of the time. DFS_ITB routing shows roughly the same traffic balancing.

Table 3 shows the average results for 30 different topologies. We observe that more benefits are obtained when using more ITBs than the ones strictly needed to guarantee deadlock-free minimal routing. UD_ITB improves over UD, on average, by factors of 1.29, 1.88, and 2.57 for 16, 32, and 64-switch networks, respectively. In a particular 64-switch network, throughput is tripled when using UD_ITB instead of UD routing. On the other hand, throughput is increased over DFS, on average, by factors of 1.12, 1.41, and 1.63 for different network sizes.

In order to analyze the latency overhead introduced by ITBs, Table 4 shows the latency introduced by in-transit buffers for very low traffic (the worst case). We show results for 512 and 32-byte messages. For 512-byte messages we observe that, on average, the in-transit buffer mechanism slightly increases average message latency. This increase is never higher than 5 percent. The latency increase is only noticeable for short messages (32 bytes). In this case, the maximum latency increase ranges from 16.66 percent to 22.09 percent for UD_ITB. The explanation is simple. Latency depends both on traversed distance and message length. ITBs only increase the latency component that depends on the number of hops that a message takes. As messages are longer, the relative contribution of this component on total message latency is smaller. Hence, the influence of ITB overhead is also smaller. On the contrary,

TABLE 3
Factor of Throughput Increase when Using In-Transit Buffers on the UD and DFS Routing

	UD_MITB vs UD			UD_ITB vs UD			DFS_ITB vs DFS		
Sw	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
16	1.00	1.29	1.13	1.00	1.57	1.29	1.01	1.20	1.12
32	1.16	1.72	1.46	1.50	2.14	1.88	1.25	1.56	1.41
64	1.60	2.25	1.91	2.20	3.00	2.57	1.50	1.85	1.63

Uniform distribution. Message size is 512 bytes.

the relative importance of ITB overhead for short messages is higher. Additionally, the latency penalty depends on the number of ITBs needed to guarantee deadlock freedom. This is also shown in Table 4 where the average latency penalty is lower when using ITBs with DFS or the minimum number of ITBs with UD (UD_MITB). Finally, the latency overhead incurred by ITBs is partially offset by the shorter paths allowed by the mechanism.

Table 5 shows the factor of throughput increase for the hot-spot, bit-reversal, local, and combined traffic patterns. We observe that the in-transit buffer mechanism always increases, on average, network throughput of UD and DFS routing. In particular, when the combined traffic pattern is used, UD_ITB improves over UD by factors of 1.26, 1.65, and 2.31 for 16, 32, and 64-switch networks, respectively. Also, DFS_ITB improves over DFS by factors of 1.14, 1.35, and 1.56 for 16, 32, and 64-switch networks, respectively.

6.4.3 In-transit Buffers on SMART

Smart-routing is not based on spanning trees. Moreover, its main goal is to balance network traffic. In fact, we have already seen the good traffic balancing achieved by this routing algorithm (Fig. 8c). Therefore, it seems that in-transit buffers will have little to offer to smart-routing. Fig. 11 shows the performance results for SMART and BALANCED_ITB routing for the uniform distribution of message destinations on 16 and 32-switch networks. In 64-switch networks, SMART routes were not available due to its high computation time. We observe that, for smart-routing, the in-transit buffer mechanism also increases network throughput. For a 32-switch network, BALANCED_ITB routing increases network throughput by a factor of 1.33.

Table 6 shows average results for 20 topologies (10 topologies for each network size). We observe that the in-transit buffer mechanism always improves over SMART (except for one 16-switch network where it obtains the same network throughput).

We have already shown that traffic is well balanced among all the links when SMART routing is used (see Fig. 8c). Fig. 12 shows traffic balancing among all the links for SMART and BALANCED_ITB routing at 0.03 flits/ns/switch. The obtained results are very similar in both cases. SMART routing is quite good in balancing traffic among all the links and, therefore, the in-transit buffer mechanism does not improve network throughput by balancing traffic even more.

To fully understand the better performance achieved by BALANCED_ITB routing, we focus now on network contention. For this reason, we plot the link blocked time for both routing algorithms. Blocked time is the percentage of time that the link stops transmission due to flow control. This is a direct measure of network contention. Fig. 13a and Fig. 13b show the link blocked time for a 32-switch network

TABLE 4
Percentage of Message Latency Increase for Very Low Traffic when Using In-Transit Buffers on UD and DFS Routing

		UD_MITB vs UD			UD_ITB vs UD			DFS_ITB vs DFS		
Message size	Sw	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
512	16	-0.24	0.76	0.20	1.01	2.63	1.69	0.22	0.74	0.57
512	32	0.26	2.42	1.24	2.31	4.20	3.33	0.90	1.08	0.93
512	64	-3.85	-1.03	-2.22	-1.23	1.46	0.31	0.67	1.27	1.02
32	16	0.80	3.41	2.29	8.52	16.66	10.52	1.56	5.50	3.49
32	32	2.33	7.57	5.32	13.16	18.07	16.44	6.09	7.28	6.59
32	64	1.40	5.97	3.64	11.69	22.09	16.87	6.44	8.56	7.64

Uniform distribution.

TABLE 5
Factor of Throughput Increase when Using In-Transit Buffers on the UD and DFS Routing for Different Traffic Patterns

		UD_MITB vs UD			UD_ITB vs UD			DFS_ITB vs DFS		
Distribution	Sw	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Hot-spot	16	0.99	1.17	1.04	0.99	1.21	1.10	1.00	1.17	1.05
Hot-spot	32	1.00	1.40	1.18	1.00	1.39	1.18	0.98	1.17	1.03
Hot-spot	64	1.60	2.08	1.71	1.66	2.57	2.03	1.21	1.49	1.35
Bit-reversal	16	0.94	1.44	1.16	0.87	1.81	1.17	0.79	1.27	1.03
Bit-reversal	32	1.12	2.00	1.59	1.56	2.57	1.87	1.20	1.99	1.51
Bit-reversal	64	1.74	2.99	2.05	2.21	3.50	2.76	1.46	2.20	1.78
Local	16	0.97	1.26	1.08	1.02	1.56	1.24	1.00	1.30	1.17
Local	32	1.00	1.40	1.16	1.12	1.60	1.44	1.15	1.45	1.29
Local	64	1.00	1.20	1.07	1.40	1.57	1.49	1.13	1.33	1.24
Combined	16	1.00	1.45	1.15	1.00	1.56	1.26	0.98	1.28	1.14
Combined	32	1.12	1.57	1.31	1.31	1.86	1.65	1.20	1.50	1.35
Combined	64	1.48	2.00	1.74	1.82	2.65	2.31	1.43	1.80	1.56

Message size is 512 bytes.

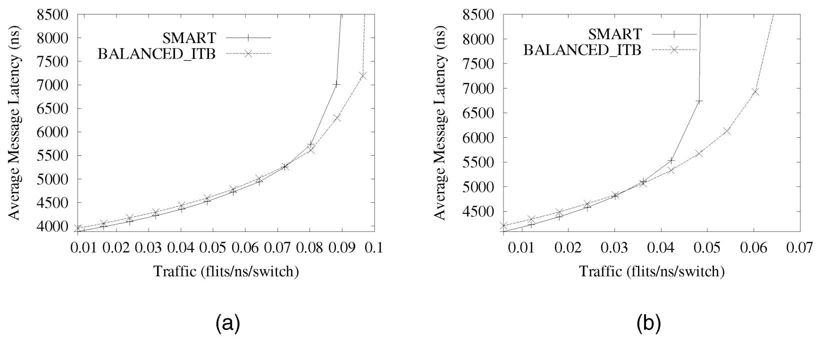


Fig. 11. Average message latency versus accepted traffic. SMART and BALANCED_ITB routing. Message size is 512 bytes. Uniform distribution. Network size is (a) 16 switches and (b) 32 switches.

for SMART and BALANCED_ITB routing, respectively. Traffic is near 0.05 flits/ns/switch. We observe that SMART has some links blocked more than 10 percent of the time, some particular links being blocked more than 20 percent of the time. On the other hand, when using in-transit buffers, blocked time is kept lower than 5 percent for all the links for the same traffic point. Fig. 13c shows the link blocked time when BALANCED_ITB routing is at its saturation point (0.06 flits/ns/switch). Thus, we observe that the BALANCED_ITB routing saturates due to network contention rather than traffic unbalancing.

Let us analyze the introduced overhead. Table 7 shows the percentage of latency increase for low traffic. For 512-byte messages, latency increase is less than 3 percent for all the topologies evaluated. For 32-byte messages, some higher penalty in latency is observed, but it is always lower than 14 percent.

For other traffic patterns, BALANCED_ITB routing also increases, on average, network throughput over SMART routing. Table 8 shows the factors of throughput increase for the hot-spot, bit-reversal, local, and combined traffic patterns. When the combined traffic pattern is used, BALANCED_ITB routing increases, on average, network throughput by a factor of 1.14 for 32-switch networks.

As a final conclusion of this section we conclude that, by using in-transit buffers on UD and DFS routing, network throughput is increased. As network size increases, higher improvements are obtained because in-transit buffers avoid congestion near the root switch, always providing deadlock-free minimal paths and balancing network traffic. Also, when applied to SMART, higher throughput is achieved. In this case, improvements are achieved by the inherent property of the mechanism in lowering network contention. On the other hand, average message latency is slightly increased. However, this increase is only noticeable for short messages and small networks. Therefore, our mechanism increases throughput over all routing algorithms for networks with source routing.

TABLE 6
Factor of Throughput Increase when Using In-Transit Buffers on SMART Routing

	BALANCED_ITB vs SMART		
Sw	Min	Max	Avg
16	1.00	1.16	1.07
32	1.11	1.33	1.23

Uniform distribution. Message size is 512 bytes.

6.5 Simulation Results: A New Routing Algorithm

Now, we focus on the new routing algorithm that will be specifically designed taking into account ITBs. First, we evaluate the load balancing feature of our routing algorithm (the standard deviation method described before). Later, we focus on reducing network contention by using different quantities of ITBs.

6.5.1 Load Balancing Behavior

In order to isolate the benefits achieved only by the traffic balancing method, we use the minimum number of ITBs that guarantee deadlock freedom (cycles are searched and broken by ITBs). This routing algorithm will be referred to as BMIN (paths are *B*alanced and *MIN*imal ITBs are used). We will compare it with the best routing algorithm evaluated in the previous section (BALANCED_ITB). In order to differentiate both routing algorithms, BALANCED_ITB will be now referred to as BSMART (paths balanced the same way as in the smart-routing algorithm). Also, as a reference, we will use a random traffic balancing algorithm in order to measure the benefits of our proposed traffic balancing algorithm. This algorithm will compute all the possible minimal paths and will select the final set of paths randomly. This routing algorithm will also use the minimum number of in-transit buffers and will be referred to as RANDOM.

Fig. 14 shows the performance results for the different routing algorithms using 16, 32, and 64-switch networks. Message size is 512 bytes. Uniform distribution of message destinations is used. Notice that the BSMART routing algorithm is not shown for 64-switch networks as it was not available.

We observe that BMIN and BSMART routing achieve a higher network throughput than the one achieved by the RANDOM routing for all network sizes. Fig. 15a and Fig. 15b show the link utilization when traffic is 0.066 flits/ns/switch in a 32-switch network for the BSMART and BMIN routing algorithms, respectively. At this traffic point, they are reaching saturation. Links are sorted by utilization. We observe that the traffic balance achieved by BSMART (Fig. 15a) is better than the one achieved by BMIN (Fig. 15b). However, this better balancing does not have an important

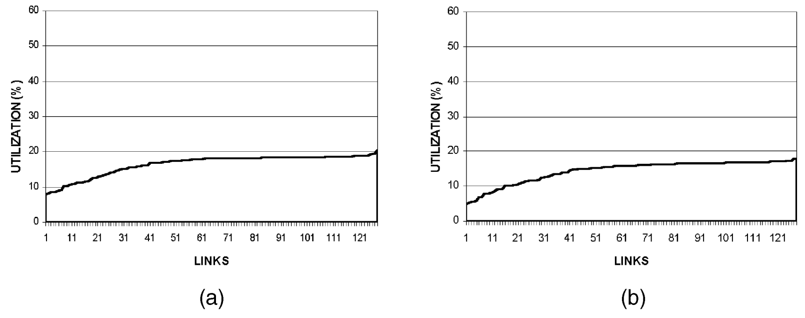


Fig. 12. Link utilization. Network size is 32 switches. Traffic is 0.03 flits/ns/switch. Message size is 512 bytes. Uniform distribution. (a) SMART and (b) BALANCED_ITB routing.

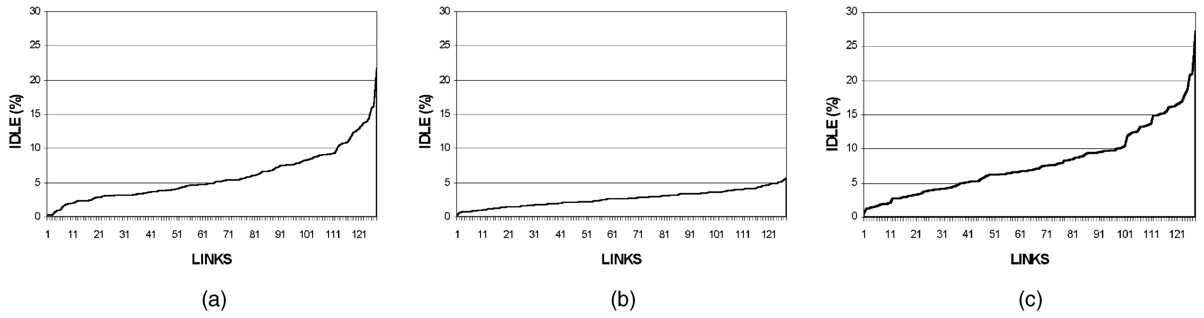


Fig. 13. Blocked Time. Network size is 32 switches. Message size is 512 bytes. Uniform distribution. Traffic is (a), (b) 0.05 flits/ns/switch and (c) 0.06 flits/ns/switch. (a) SMART and (b), (c) BALANCED_ITB.

impact on network performance (as shown in Fig. 14). Moreover, BSMART shows a slight decrease in performance with respect to BMIN routing algorithm. This behavior is due to network contention. BSMART uses, on average, 0.32 ITBs per message, whereas BMIN uses a bit more, 0.38. BMIN uses more ITBs because it introduces more cycles in the CDG.

The main difference between both routing algorithms is that BSMART uses a linear programming solver when computing paths, whereas our routing algorithm uses an iterative process that is faster. This is an important issue in systems that are prone to changes in network topology (i.e., COWs).

Fig. 15c shows the link utilization for the RANDOM routing algorithm when is reaching saturation (0.053 flits/ns/switch) using the same network topology (32 switches). We can see that it achieves a worse traffic balance than BMIN and BSMART. This difference affects the final performance, as Fig. 14b shows. In particular, BMIN and BSMART increase RANDOM throughput by a factor of 1.3. This result indicates that an effort must be made in balancing network traffic. With a simple and easy traffic balancing algorithm, good results are achieved, with additional efforts (BSMART) yielding no further improvements.

Table 9 shows average results for other randomly generated topologies. It shows the minimum, maximum, and average factor of throughput increase when using the BMIN routing algorithm with respect to the BSMART and RANDOM routing algorithms, respectively. With respect to the BSMART routing algorithm, BMIN achieves nearly the same throughput. Therefore, with an easy traffic balancing algorithm (BMIN), we can achieve the performance obtained by a sophisticated and time-consuming load balancing algorithm (BSMART). The BMIN algorithm requires less computation time than BSMART, even when considering the entire set of possible paths, for all graphs considered in our analysis.

TABLE 7
Percentage of Message Latency Increase for Very Low Traffic when Using In-Transit Buffers on SMART Routing

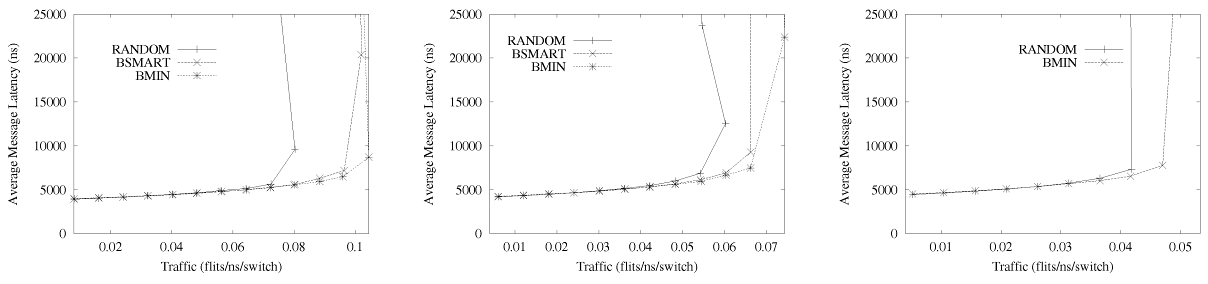
		BALANCED_ITB vs SMART		
Message size	Sw	Min	Max	Avg
512	16	-0.20	2.22	1.64
512	32	1.78	2.95	2.34
32	16	-0.35	10.18	7.77
32	32	9.26	13.28	11.00

Uniform distribution.

TABLE 8
Factor of Throughput Increase when Using In-Transit Buffers on SMART Routing for Different Traffic Patterns

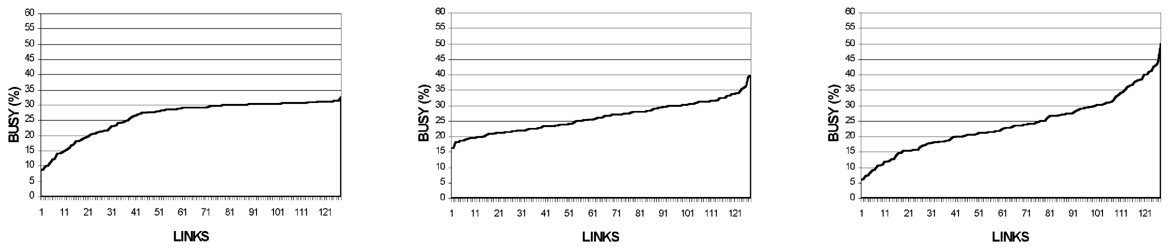
		BALANCED_ITB vs SMART		
Distribution	Sw	Min	Max	Avg
Hot-spot	16	0.85	1.17	0.96
Hot-spot	32	1.00	1.00	1.00
Bit-reversal	16	0.73	1.13	0.93
Bit-reversal	32	0.99	1.45	1.21
Local	16	1.00	1.17	1.10
Local	32	1.10	1.29	1.17
Combined	16	1.00	1.17	1.06
Combined	32	1.04	1.27	1.14

Message size is 512 bytes.



(a) (b) (c)

Fig. 14. Average message latency versus accepted traffic. BSMART, BMIN, and RANDOM routing. Uniform distribution. Message size is 512 bytes. Network size is (a) 16 switches, (b) 32 switches, and (c) 64 switches.



(a) (b) (c)

Fig. 15. Link utilization. Network size is 32 switches. Message size is 512 bytes. Traffic is (a), (b) 0.066 and (c) 0.053 flits/ns/switch. (a) BSMART, (b) BMIN, and (c) RANDOM.

Also, with respect to the RANDOM routing algorithm, BMIN always improves network throughput (on average by a factor of 1.21 for 16-switch networks). However, as network size increases, improvements are, on average, the same. This is because the RANDOM routing algorithm is fully scalable (always uses minimal paths) and the only difference with respect to BMIN is the balancing algorithm.

The main drawback of using ITBs is the added latency to messages. However, this penalty is only noticeable in low network traffic loads and especially in short messages [9]. For medium and high network traffic loads, the better traffic handling hides the higher latency of messages that use ITBs. Table 10 shows the percentage of latency increase when using the BMIN routing algorithm with respect to the UD, SMART, BSMART, and RANDOM routing algorithms, respectively, in low traffic loads (UD and SMART routings are included to see the latency penalty over routings that do not use ITBs). Message sizes of 32 and 512 bytes are used. Also, a combination of both message sizes are considered under the term bimodal (70 percent of messages are 32-bytes long, whereas 30 percent are 512-byte long). We can see that the latency penalty is higher than 10 percent only when short messages are used on medium and large networks (32 and 64 switches) and with respect to the UD and SMART routing. However, when traffic with different message sizes (bimodal) is used, the average latency penalty incurred over UD and SMART is significantly reduced (under 9 percent for all networks).

For the rest of the routing algorithms, the latency increase remains under 10 percent for all networks and all message sizes. These routing algorithms also use ITBs and, therefore, all of them expose the same latency penalty.

As stated earlier, the load balancing algorithms usually rely on an expected traffic pattern (usually the uniform one), but different traffic patterns can appear in the network. To investigate this, we also evaluate the routing algorithms under different traffic patterns. Table 11 shows the factor of throughput increase when using the BMIN routing algorithm with respect to the other routing algorithms under different traffic patterns.

With the hot-spot and bit-reversal traffic patterns, BMIN achieves nearly the same throughput as BSMART. Also, when comparing with RANDOM, BMIN achieves smaller benefits (7 percent and 12 percent on average, respectively, for 32-switch networks).

For the local traffic pattern, nearly the same throughput is achieved with all the routing algorithms. The important issue here is that the ITB mechanism does not decrease network throughput with a local traffic pattern.

Finally, with a combined traffic pattern, the BMIN routing algorithm obtains nearly the same benefits as described for the uniform traffic pattern. As the traffic pattern changes, path lengths change. With ITBs, the longer the path, the greater the improvement over alternative routing algorithms. Also, as path length is reduced (local traffic pattern), fewer

TABLE 9
Factor of Throughput Increase when Using BMIN with Respect to BSMART and RANDOM

	BSMART			RANDOM		
Sw	Min	Max	Avg	Min	Max	Avg
16	0.95	1.13	1.03	1.07	1.39	1.21
32	0.99	1.12	1.03	1.08	1.27	1.18
64	N/A	N/A	N/A	1.00	1.27	1.16

Uniform distribution. Message size is 512 bytes.

TABLE 10
Percentage of Latency Increase when Using BMIN with Respect to UD, SMART, BSMART, and RANDOM

		BMIN vs UD			BMIN vs SMART			BMIN vs BSMART			BMIN vs RANDOM		
Sw	Msgs	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
16	32	2.27	10.92	6.83	9.95	12.77	8.04	-2.91	3.92	0.25	0.27	7.05	3.13
32	32	12.45	19.15	15.22	12.43	18.58	15.57	0.40	6.56	4.12	-0.83	6.96	4.23
64	32	13.95	20.09	17.18	N/A	N/A	N/A	N/A	N/A	N/A	1.16	6.37	4.31
16	512	0.06	1.65	0.81	0.07	2.44	1.63	-0.96	0.70	0.00	-0.17	1.15	0.48
32	512	2.32	4.56	3.22	2.35	3.97	3.38	0.30	1.61	1.02	-0.17	1.59	0.94
64	512	-0.14	2.83	1.76	N/A	N/A	N/A	N/A	N/A	N/A	0.43	1.68	1.12
16	bimodal	0.70	3.92	2.00	0.98	5.61	3.69	-2.40	1.84	-0.08	-0.70	3.23	1.11
32	bimodal	3.89	8.90	5.91	6.35	9.86	8.23	0.14	3.97	2.27	-0.93	4.02	1.93
64	bimodal	-1.42	4.69	2.95	N/A	N/A	N/A	N/A	N/A	N/A	0.90	5.28	2.79

Uniform distribution.

ITBs are needed and, therefore, throughput of the different routing algorithms will be very similar.

6.5.2 Network Contention Behavior

Now, we focus on the second part of the routing algorithm (network contention control). For this, we proposed four algorithms to assign ITBs in the network while breaking cycles (see Section 5.2). Each of them use different numbers of ITBs. We will refer to them as B33, B50, B66, and BMAX. Also, we will compare them with the BMIN routing evaluated before. Notice that BMIN, B33, B50, B66, and BMAX use the new load balancing algorithm and the main difference between them is the number of ITBs used: the minimum possible, 33 percent, 50 percent, 66 percent, and the maximum possible, respectively.

Fig. 16 shows the performance results for the different routing algorithms using 512-byte messages and a uniform traffic pattern.

We observe that, for the 16-switch topology, the worst routing algorithms (in terms of network throughput) are UD and SMART. However, the best routing algorithm is BMAX, followed by B66 and B50. Finally, B33 achieves better network throughput than BMIN. Therefore, we can conclude that, as more ITBs are used, a higher network throughput is achieved. BMAX increased BMIN network throughput by a factor of 1.27. However, it is worth to say that differences in network throughput are short. In terms of message latencies, we can see that the use of more ITBs increases average message latency.

For 32 and 64-switch networks (Fig. 16b and Fig. 16c), we obtain similar conclusions. As we can see, network throughput reached by B50, B66, and BMAX is practically the same. Therefore, the increase in network throughput is not directly related to the number of ITBs. With a moderated number of ITBs (B50) we can reach roughly the same network throughput as with the maximum number of ITBs (BMAX) with a lower latency penalty. Finally, we can see that BMIN network throughput is increased by BMAX in factors of 1.36 y 1.5 in 32 and 64-switch networks, respectively.

By using more ITBs, routings reduce network contention. Fig. 17 shows the percentage of blocked time of links using different routing algorithms. Network size is 32 switches and traffic is 0.066 flits/ns/switch (where BMIN is reaching saturation).

We can see how, as the number of ITBs is increased, network links exhibit a lower blocking time. By using BMIN (Fig. 17a), there are network links blocked more than 14 percent of time. These links contribute to overall network saturation. When we put more ITBs in the network with the B50 routing (Fig. 17c), all links are blocked less than 10 percent of the time. The more ITBs we put, the better. BMAX (Fig. 17e) is the extreme case where ITBs are put in all the switches. We observe that 95 percent of the links are blocked less than 7 percent of the total time. Therefore, the ITB mechanism also helps in increasing network throughput by reducing network contention. But, on the other hand, the latency penalty (studied later) should be taken into account. The lower the number of ITBs used, the lower this penalty. We observe that the B50 routing algorithm reaches nearly the same network throughput as BMAX by using half the ITBs with respect to BMAX.

For other network topologies, similar results have been obtained. Table 12 shows the minimum, maximum, and average factors of throughput increase for different network sizes and different routing algorithms. On average, B50 increases network throughput of BMIN by factors of 1.18, 1.27, and 1.35. Regarding the classical routing algorithms (UD and SMART), the improvement reaches, in some cases, on average, factors of 3.77. We also observe that the B50 routing algorithm achieves roughly the same network throughput as BMAX.

TABLE 11
Factor of Throughput Increase when Using BMIN with Respect to BSMART and RANDOM Routings for Different Traffic Patterns

		BMIN vs BSMART			BMIN vs RANDOM		
Sw	Pattern	Min	Max	Avg	Min	Max	Avg
16	Hot-spot	0.99	1.02	1.00	0.99	1.17	1.02
32	Hot-spot	0.99	1.15	1.05	0.99	1.15	1.07
64	Hot-spot	N/A	N/A	N/A	0.87	1.15	1.02
16	Bit-reversal	0.94	1.50	1.09	0.76	1.59	1.18
32	Bit-reversal	0.80	1.24	1.05	0.88	1.51	1.12
64	Bit-reversal	N/A	N/A	N/A	0.92	1.20	1.07
16	Local	0.96	1.10	1.04	0.99	1.15	1.06
32	Local	1.00	1.11	1.07	0.95	1.10	1.04
64	Local	N/A	N/A	N/A	0.96	1.09	1.03
16	Combined	0.99	1.09	1.02	1.08	1.32	1.16
32	Combined	1.00	1.13	1.06	1.03	1.37	1.17
64	Combined	N/A	N/A	N/A	0.96	1.26	1.11

Message size is 512 bytes.

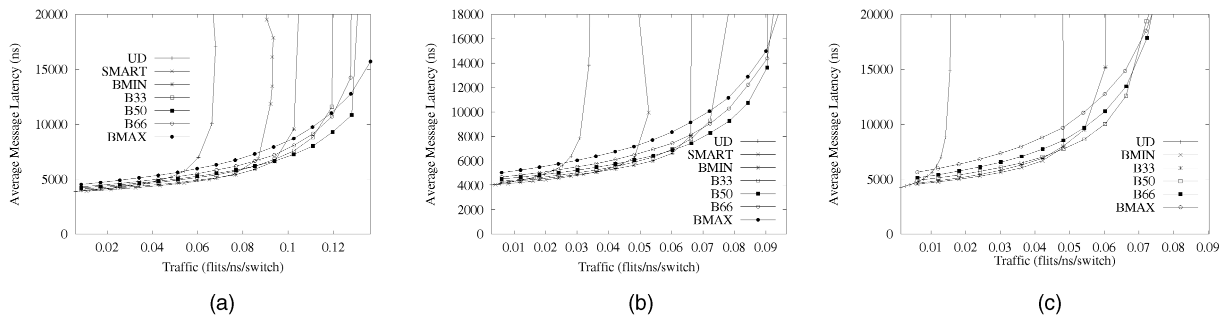


Fig. 16. Average message latency versus accepted traffic. Uniform distribution. Message size is 512 bytes. Network size is (a) 16 switches, (b) 32 switches, and (c) 64 switches.

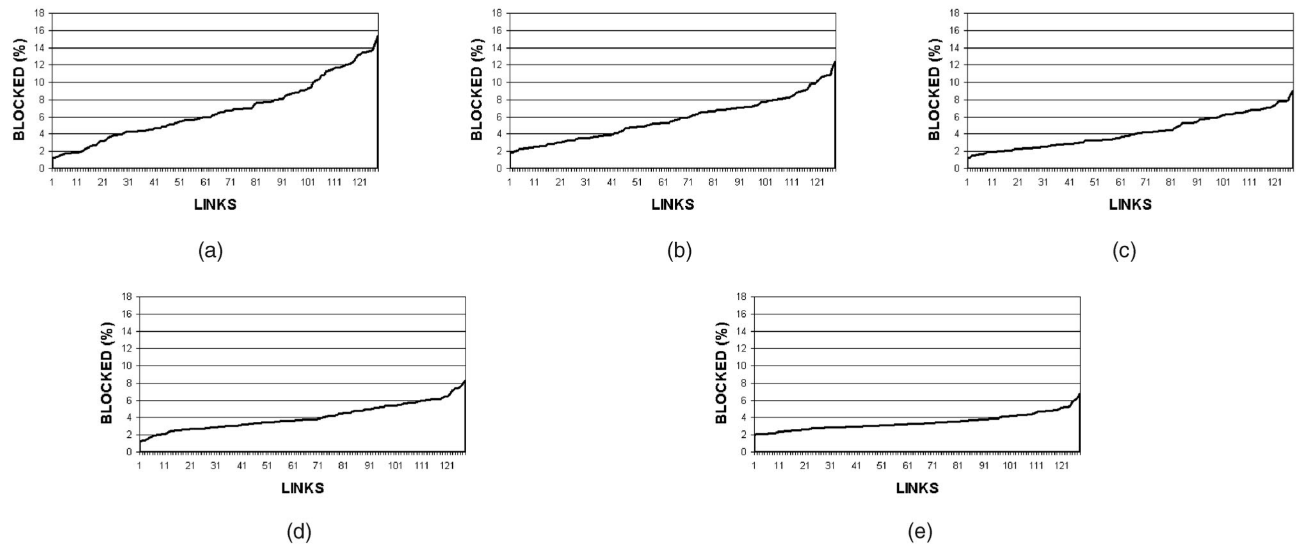


Fig. 17. Blocked time (network contention). Traffic is 0.066 flits/ns/switch. Network size is 32 switches. Message size is 512 bytes. Uniform distribution. (a) BMIN, (b) B33, (c) B50, (d) B66, and (e) BMAX routing.

TABLE 12
Factor of Throughput Increase when Using B50 with Respect to UD, SMART, BMIN, and BMAX Routings for the Uniform Distribution

	B50 vs UD			B50 vs SMART			B50 vs BMIN			B50 vs BMAX		
Sw	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
16	1.35	2.03	1.70	1.05	1.46	1.32	0.97	1.37	1.18	0.91	1.00	0.98
32	2.10	2.81	2.51	1.56	1.82	1.69	1.13	1.36	1.27	0.94	1.07	0.99
64	3.29	4.35	3.77	N/A	N/A	N/A	1.22	1.45	1.35	0.93	1.09	1.04

Message size is 512 bytes.

Because these routing algorithms (B33, B50, B66, and BMAX) use many more ITBs than others (BMIN), the latency penalty has to be evaluated. Table 13 shows the percentage of latency increase when using the B50 routing algorithm instead of the UD, SMART, BMIN, and BMAX, respectively. We observe that, by using many more ITBs, as B50 does, the latency penalty is increased, on average, up to 50 percent with respect to UD and SMART. But, this happens only for short messages and in low traffic conditions. For bimodal traffic, the average latency penalty is 20 percent on average. Although this penalty should be taken into account, the great improvements in network throughput should also be considered.

For the sake of brevity, results for other traffic patterns are not shown. However, the study has been performed showing that, for bit-reversal, hot-spot, and combined traffic patterns, throughput improvements are lower (but still noticeable). For local traffic pattern, throughput is never decreased.

In summary, load balance is not the only key contributor to network performance. Reducing network contention with ITBs can yield significant improvements in network throughput. The B50 routing algorithm exploits both issues and outperforms previous proposals.

7 Conclusions

In previous papers, we proposed the in-transit buffer mechanism (ITB) to improve network performance in networks with source routing. Although the mechanism was primarily intended for supplying minimal paths in the up*/down* routing algorithm by removing *down* $\rightarrow$ *up*

TABLE 13
Percentage of Latency Increase when Using B50 with Respect to UD, SMART, BMIN, and BMAX Routings for the Uniform Distribution

		B50 vs UD			B50 vs SMART			B50 vs BMIN			B50 vs BMAX		
Sw	Msgs	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
16	32	29.61	56.61	35.24	30.42	54.59	36.78	18.81	53.13	26.60	-27.12	-23.12	-25.00
32	32	42.57	51.07	46.29	42.60	53.52	46.74	22.48	33.03	26.96	-31.42	-27.49	-29.62
64	32	45.30	51.29	48.33	N/A	N/A	N/A	23.96	28.84	26.58	-36.92	-34.47	-35.84
16	512	6.08	13.27	7.92	7.16	13.28	8.80	5.06	13.19	7.05	-8.22	-6.58	-7.45
32	512	9.12	11.45	9.91	8.84	11.73	10.09	5.36	8.01	6.48	-11.01	-9.58	-10.25
64	512	8.58	11.03	9.55	N/A	N/A	N/A	7.03	8.57	7.66	-13.55	-12.18	-13.02
16	bimodal	11.80	25.77	15.86	14.07	26.12	17.77	8.59	25.54	13.49	-16.73	-13.61	-15.08
32	bimodal	17.40	24.09	20.21	19.91	27.07	22.84	14.11	19.22	16.07	-22.65	-16.78	-20.65
64	bimodal	14.51	20.94	18.53	N/A	N/A	N/A	N/A	N/A	N/A	-31.63	-22.49	-26.26

channel dependencies, we have found that it also serves as a powerful mechanism to reduce network contention and to obtain a better network traffic balance. Moreover, it can be applied not only to up*/down*, but to any source-based routing algorithm.

In this paper, we apply the ITB mechanism to up*/down*, DFS, and smart-routing schemes, analyzing its behavior in detail. Also, we have presented a new source-based routing algorithm. This routing algorithm computes a set of minimal paths among all the network nodes. Then, it chooses a subset of them that offers good traffic balance through a very simple and low time consuming algorithm. This algorithm is based on the computation of the standard deviation of the number of paths that cross each network link. Finally, it guarantees deadlock freedom and also reduces network contention by a clever ITB allocation procedure.

Results show that, in general, the in-transit buffer mechanism improves network performance for all the analyzed source routing algorithms. We already knew that up*/down* routing was significantly improved due to the high number of routing restrictions imposed by this routing algorithm and the unbalanced traffic nature of the spanning trees. In this paper, we have demonstrated that, by using ITBs, improved source routing algorithms, like DFS and smart-routing, can also take advantage of ITBs. The improvement can be as high as 85 percent for DFS for a 64-switch network and 33 percent for smart-routing for a 32-switch network, for the uniform distribution of message destinations.

On the other hand, the new routing algorithm offers very good traffic balance behavior. When compared with smart-routing with ITBs and a random traffic balancing algorithm, it achieves factors of throughput increases up to 1.12 and 1.27, respectively for 32-switch networks. Most important, it is able not only to improve smart-routing, but also to strongly reduce routing table computation overhead. In fact, tables for large networks (64 switches) were rapidly computed. Also, when the new routing algorithm is combined with a clever strategy to allocate more ITBs in the network, network contention is strongly reduced. In particular, it is able to improve network throughput to unpredecedented levels: up to 4.35, and 1.82 of factor improvements over up*/down* (64 switches) and smart-routing (32 switches), respectively, for the uniform distribution of message destinations.

ACKNOWLEDGMENTS

This work was supported by the Spanish MCYT under Grants TIC2000-1151-C07 and 1FD97-2129.

REFERENCES

[1] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J. Seizovic, and W. Su, “Myrinet—A Gigabit per Second Local Area Network,” *IEEE Micro,* pp. 29-36, Feb. 1995.

[2] R.V. Bopana and S. Chalasani, “A Comparison ofAdaptive Wormhole Routing Algorithms,” *Proc. 20th Ann. Int'l Symp. Computer Architecture,* May 1993.

[3] L. Cherkasova, V. Kotov, and T. Rokicki, “Fibre Channel Fabrics: Evaluation and Design,” *Proc. 29th Int'l Conf. System Sciences,* Feb. 1995.

[4] S. Coll, J. Flich, M.P. Malumbres, P. Lopez, J. Duato, and F.J. Mora, “A First Implementation of In-Transit Buffers on Myrinet GM Software,” *Proc. Workshop Comm. Architecture for Clusters,* Apr. 2001.

[5] W.J. Dally and C.L. Seitz, “Deadlock-Free Message Routing in Multiprocessors Interconnection Networks,” *IEEE Trans. Computers,* vol 36, no. 5, pp. 547-553, May 1987.

[6] W.J. Dally, “Virtual-Channel Flow Control,” *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 2, pp. 194-205, Mar. 1992.

[7] J. Duato, “A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks,” *IEEE Trans. Parallel and Distributed Systems,* vol. 6, pp. 1055-1067, 1995.

[8] J. Flich, M.P. Malumbres, P. López, and J. Duato, “Performance Evaluation of a New Routing Strategy for Irregular Networks with Source Routing,” *Proc. Int'l Conf. Supercomputing,* May 2000.

[9] J. Flich, P. López, M.P. Malumbres, and J. Duato, “Improving the Performance of Regular Networks with Source Routing,” *Proc. Int'l Conf. Parallel Processing,* Aug. 2000.

[10] J. Flich, P. López, M.P. Malumbres, and J. Duato, “Boosting the Performance of Myrinet Networks,” *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 7, July 2002.

[11] GM homepage, http://www.myri.com/GM', 2001.

[12] J. Kim and A. Chien, “An Evaluation of the Planar/Adaptive Routing,” *Proc. Fourth IEEE Int'l Symp. Parallel and Distributed Processing,* 1992.

[13] P.R. Miller, “Efficient Comunications for Fine-Grain Distributed Computers,” PhD thesis, Southampton Univ., 1991.

[14] Myrinet, M2-CB-35 LAN cables, http://www.myri.com/myrinet/product_list.html, 2001.

[15] S.S. Owicki and A.R. Karlin, “Factors in the Performance of the AN1 Computer Network,” *Performance Evaluation Rev.,* vol. 20, pp. 167-180, June 1992.

[16] W. Qiao and L.M. Ni, “Adaptive Routing in Irregular Networks Using Cut-Through Switches,” *Proc. 1996 Int'l Conf. Parallel Processing,* Aug. 1996.

[17] J.C. Sancho, A. Robles, and J. Duato, "New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks," *Proc. Workshop Comm. and Architectural Support for Network-Based Parallel Computing,* Jan. 2000.

[18] M.D. Schroeder et al., "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," Technical Report SRC research report 59, DEC, Apr. 1990.

[19] S.L. Scott and J.R. Goodman, "The Impact of Pipelined Channels on k-Ary n-Cube Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 1, pp. 2-16, Jan. 1994.

[20] R. Sheifert, *Gigabit Ethernet.* Addison-Wesley, Apr. 1998.

[21] F. Silla and J. Duato, "Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology," *Proc. 1997 Int'l Conf. High Performance Computing,* Dec. 1997.

[22] F. Silla, M.P. Malumbres, A. Robles, P. López, and J. Duato, "Efficient Adaptive Routing in Networks of Workstations with Irregular Topology," *Proc. Workshop Comm. and Architectural Support for Network-Based Parallel Computing,* Feb. 1997.

José Flich received the MS and PhD degrees in computer science from the Technical University of Valencia (Universidad Politecnica de Valencia), Spain, in 1994 and 2001, respectively. He joined the Department of Computer Engineering (DISCA), Universidad Politécnica de Valencia in 1998, where he is currently an associate professor of computer architecture and technology. His research interests are related to high performance interconnection networks for multiprocessor systems and clusters of workstations. He is a member of the IEEE.

Pedro López received the BEng degree in electrical engineering and the MS and PhD degrees in computer engineering from the Technical University of Valencia (Universidad Politecnica de Valencia), Spain, in 1984, 1990 and 1995, respectively. He joined the Department of Computer Engineering (DISCA), Universidad Politécnica de Valencia in 1986, where he is currently a professor of computer architecture and technology. He has taught several courses on computer organization and computer architecture. His research interests include high performance interconnection networks for multiprocessor systems and clusters of workstations. He is a member of the editorial board of *Parallel Computing* and a member of the IEEE Computer Society.

Manuel Perez Malumbres received the MS and PhD degrees in computer engineering from the Technical University of Valencia (UPV), Spain, in 1991 and 1996, respectively. He is currently an associate professor in the Computer Engineering Department (DISCA) at the UPV and his research and teaching activities are related to multimedia networking and high-speed networks. He is a member of the IEEE.

José Duato received the MS and PhD degrees in electrical engineering from the Technical University of Valencia, Spain, in 1981 and 1985, respectively. He is currently a professor in the Department of Computer Engineering (DISCA), Technical University of Valencia (Universidad Politecnica de Valencia), Spain, and adjunct professor in the Department of Computer and Information Science, Ohio State University. He is currently researching multiprocessor systems, networks of workstations, interconnection networks, and multimedia systems. His theory on deadlock-free adaptive routing has been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E router, and the router embedded in the new Alpha 21364 microprocessor. He coauthored the text *Interconnection Networks: An Engineering Approach* with S. Yalamanchili and L.M. Ni. Dr. Duato served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and *IEEE Transactions on Computers*. He has also been or is a member of the program committees for several major conferences (ICPADS, ICDCS, Europar, HPCA, ICPP, MPPOI, HiPC, PDCS, ISCA, IPPS/SPDP, ISPAN). He was the general cochair for ICPP 2001. He is a member of the IEEE and the IEEE Computer Society.

Tomas Rokicki received the BSEE degree from Texas A&M University in 1985 and the PhD degree in computer science from Stanford University in 1994. Despite years of research in fields such as circuit verification and networking, his most noted accomplishment is the program dvips. When not running in the wonderful California weather, he works as Director of Technology for Instantis.