Fully Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations

Gustavo D. Pifarré, Luis Gravano, Sergio A. Felperin, and Jorge L.C. Sanz, *Fellow, IEEE*

Abstract— **This paper deals with the problem of packet-switched routing in parallel machines. Several new routing algorithms for different interconnection networks are presented. While the new techniques apply to a wide variety of networks, routing algorithms will be shown for the hypercube, the two-dimensional mesh, and the shuffle-exchange. Although the new techniques are designed for packet routing, they can be used alternatively for virtual cut-through routing models. The techniques presented for hypercubes and meshes are fully-adaptive and minimal. A fully-adaptive and minimal routing is one in which *all* possible minimal paths between a source and a destination are of potential use at the time a message is injected into the network. Minimal paths followed by messages ultimately depend on the local congestion encountered in each node of the network. In the shuffle-exchange network, the routing scheme also exhibits adaptivity but paths could be up to $3 \log N$ long for an N node machine. The shuffle-exchange algorithm is the first adaptive and deadlock-free method that requires a small (and independent of N) number of buffers and queues in the routing nodes for that network.**

Furthermore, all of the new techniques are completely free of deadlock situations. In dynamic message injection models, the routing methods are also ensured to be free of livelock if messages competing for resources are handled with fairness. In contrast to other approaches in which adaptivity, deadlock and livelock freedom can be guaranteed at the expense of complex architectures, the algorithms presented in this paper require a very moderate amount of routing resources. In particular, it will be shown that only two central queues per routing node of the network are necessary for the cases of the two-dimensional mesh and the hypercube, and four queues for the shuffle-exchange.

Simulations are reported showing the performance of the routing algorithms for two-dimensional meshes and hypercubes for different traffic models: random, complement, transpose, bit-reversal and leveled permutations. The performance of the routing algorithms is measured in terms of throughput, maximum and average latency, and saturation point. In the case of the mesh network, the new method is compared to an oblivious scheme similar to the x-y or e-cube router. Simulation results are reported for hypercubes up to 16-K nodes and for meshes of 1-K nodes. These results demonstrate that the new algorithms outperform oblivious e-cube-type techniques.

Manuscript received July 25, 1991; revised June 16, 1992.

G. D. Pifarré and S. A. Felperin are with Advanced Solutions Group, IBM Argentina, Ing. E. Butti 275, (1300) Buenos Aires, Argentina; and the Computer Science Dept., IBM Almaden Research Center, San José, CA 95120–6099.

L. Gravano is with the Department of Computer Science, Stanford University, Stanford, CA 94305–2140.

J. L. Sanz is with the Coordinated Science Lab and Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, and with IBM Argentina.

IEEE Log Number 9214462.

Index Terms—**Massively parallel computing, parallel communication, message routing algorithms, mesh-connected computer, hypercube, shuffle-exchange.**

I. Introduction

MESSAGE routing in large interconnection networks has attracted a great deal of interest in recent years. Different underlying machine models have been used and proposed [1]–[13]. Some fundamental distinctions among routing algorithms involve the length of the messages injected in the network, the static or dynamic nature of the injection model, special assumptions on the semantic of the messages, architecture of the network and router, degree of synchronization in the hardware, and others.

In terms of message length, several issues have been studied concerning the ways to handle long messages (of potentially unknown size) and very short messages (typically of 150–300 bits). In packet-switching routing, the messages are of constant (and small) size, and they are stored completely in every node they visit. In [14], a survey of some packet routing algorithms has been presented. In [15], simulation results have been shown comparing a number of different oblivious packet routing schemes on the hypercube. In worm-hole routing [1], messages of unknown size are routed in the network. These messages are never stored completely in a node. Only pieces of the messages, called flits, are buffered when routing. For a review of recent worm-hole methods, see [16]. In between packet-routing and worm-hole lie some hybrid approaches. In these methods [17], a message is routed by using a worm-hole technique until it gets blocked in a node by traffic. In this case, the message is buffered completely in the node, if buffers are large enough with respect to message length.

Two subjects of long-standing interest in routing are deadlock and livelock freedom. Techniques that perform without deadlocks or livelocks have been shown on different models. Some algorithms succeed in accomplishing deadlock-free or livelock-free routing only in a probabilistic sense [7], [18]. In other algorithms, deadlock freedom is guaranteed in a deterministic sense [19], [20]. Several techniques achieve this by defining an ordering on the critical resources, and allowing each message to progress throughout the network by occupying resources in a strictly monotonic fashion [1]–[3], [21]–[24], among others. This idea results in the generation of a directed acyclic graph (DAG) of the resources.

A desirable feature of routing algorithms is adaptivity, i.e., the ability of messages to use alternative paths toward their destinations according to traffic congestion in the nodes of the network. Recent simulation results for k-ary n-cubes have shown that adaptivity improves the performance of dynamic injection routing when compared to oblivious methods [25]. However, finding deterministic and probabilistic bounds for static models of message injection in adaptive routing is still an open problem for all cube-type networks.

Restricting the set of available paths in the network to a subset suitably chosen is a common way to reduce the hardware resources necessary for deadlock-free routing. When stringent restrictions are applied, oblivious algorithms or methods with partial adaptivity will be obtained. In [21], an adaptive method for routing in the hypercube is proposed. In the work of [1], practical oblivious routing techniques for a variety of networks are presented. Oblivious algorithms have been studied thoroughly for meshes and tori [26]. In [26], the performance of these algorithms for meshes has been analyzed, for both static and dynamic injection models, and for both packet switching and virtual cut-through techniques. Recently, some mathematical analyses have been reported on the performance of worm-hole oblivious algorithms [27]. On the other hand, if few restrictions are imposed on the set of possible routes generated by a routing function, impractical algorithms may result. For example, the structured buffer pool [23], [24] guarantees deadlock freedom by adding all necessary resources so that a DAG is obtained. This will result in an excessive amount of hardware necessary in a routing node and this situation will not be improved by allowing messages to depart from the DAG routes if queue space is available [24].

A *fully-adaptive minimal* routing scheme is one in which all possible minimal paths between a source and a destination are of potential use at the time messages are injected into the network. Paths followed by the messages depend on the traffic congestion found in the nodes of the network. For example, the minimal routing functions presented in [21] and [22] are *not* fully-adaptive because several minimal routes are not allowed to take place. Full-adaptivity is a feature from which one can hope to obtain the best possible performance if no source of randomization is used. Full-adaptivity has been used by Upfal in [4] to produce a deterministic optimal algorithm for routing in the multibutterfly. Multibutterflies are extremely rich in terms of the number of minimal paths between any pair of nodes. Recently, the principles shown in [28] and this paper have been used for the same routing model in n-dimensional torus networks by using three queues per node [29], and this result is optimal for the given model and network [30].

The recent work reported in [7] shows a striking reduction of hardware resources by providing an adaptive deadlock-free routing algorithm dubbed *Chaos*. The method has a non-zero probability that a message will not reach its destination after t routing steps, for an arbitrary t. However, this probability tends to zero as t approaches infinity. Furthermore, the technique in [7] and [31] applies only to packet and virtual cut-through routing and paths followed by the messages are not necessarily minimal.

Lately, new algorithms for worm-hole routing have been reported [25], [32], and [33]. In [25], a method for deadlock-free adaptive routing in k-ary n-dimensional cubes is presented such that it can be used for worm-hole routing. The new technique is based on a dynamic view of the conditions under which deadlock may arise. Routing of messages is accomplished by enforcing certain priorities on the use of virtual channels potentially intervening in deadlock conditions. In [32], a technique based on the use of multiple independent "lanes" associated with each physical link in a routing node is presented. Given a fixed amount of storage space allocated to each physical channel, it is shown that breaking the storage into several buffers is a convenient methodology for improving network performance. Simulations for worm-hole routing on a multistage interconnection network are shown. On the other hand, in [33], new algorithms for deadlock-free fully-adaptive minimal routing are presented. These algorithms are for k-ary n-dimensional cubes and n-dimensional mesh-connected networks. The techniques require 12 virtual channels for some physical links in the two-dimensional torus and a total of 36 virtual channels per node. In the 3-D mesh, up to 8 virtual channels for some physical links are necessary and a total of 32 per routing node. In [34], a fully-adaptive minimal worm-hole routing technique for the bidimensional torus network was shown, requiring 8 virtual channels per bidirectional link. In [35], fully-adaptive worm-hole algorithms were introduced for a variety of networks. For example, a routing algorithm for n-dimensional tori was presented requiring just 5 virtual channels per bidirectional link in all but the most significant dimension, in which only 3 virtual channels are needed [36].

In this paper, a number of algorithms for packet routing are shown. These techniques are fully-adaptive minimal (except for the one for the shuffle-exchange), deadlock- and livelock-free and require a very moderate amount of resources in the routing nodes. The new methods are presented for hypercubes, meshes, and shuffle-exchange networks. These results have been originally presented in [28].

The family of algorithms presented in this paper differs in a radical way from deflection routing, hot-potato routers, and other fully-adaptive techniques that *misroute* messages to ensure deadlock freedom [37], [38]. Hot-potato routers are non-minimal and furthermore, messages can get misrouted in the presence of conflicts in the use of the output ports in a node. The misrouting mechanism is necessary for deflection or hot-potato routers to be free of deadlock situations. On the other hand, misrouting creates potential livelock in dynamic routing as a message may take arbitrarily long time to arrive at its destination.

The organization of this paper is as follows. In Section II, some terminology and concepts concerning static and dynamic deadlock freedom will be introduced. In Sections III, IV, and V, the main results of this paper are presented. In these sections, algorithms for fully-adaptive routing on hypercubes, two-dimensional meshes, and for adaptive routing on shuffle-exchange networks will be shown. In Section VI, the functional designs of the routing node for the above three interconnections are shown. These designs give empha-

sis to the number of buffers sharing a physical link, and the operation and number of central queues in the node. In Section VII, the results obtained from the simulations involving two-dimensional meshes and hypercubes are presented. These simulation results have been originally reported in [28] and [34]. In Section VIII, some conclusions are presented.

II. Definitions and Terminology

In the packet routing model used in this paper, the critical resources are the queues used to store the messages. This model has been used in a number of routing papers [21], [29], [30], [7], [31]. *Deadlock* will arise, if and only if there exists a set of full queues occupied by messages such that all of these messages need a slot of a queue that belongs to the set in order to continue their way toward their destinations. *Livelock* takes place whenever a message can keep moving infinitely many times without getting delivered. This phenomenon may happen when the routing technique is nonminimal, i.e., when messages can take paths that are longer than the shortest paths.

Each node of the network will have associated with it a certain number of queues. Each node has a pair of distinct queues, namely the injection and the delivery queues. Messages will be injected in the injection queue, and they will be consumed from the delivery queue. The set of injection queues of all the network will be referred to as $InjectQ$, and the set of delivery queues, as $DelivQ$. The routing function will be expressed in terms of the queues of each node. Every delivery queue identifies a unique node of the network. Each message has a destination associated with it, given by the function $Dest$: Messages $\rightarrow DelivQ$.

A total routing function $\mathcal{R} : Queues \times DelivQ \rightarrow \mathcal{P}(Queues)$, where $\mathcal{P}(Queues)$ is the powerset of $Queues$, is such that $\mathcal{R}(q,d)$ indicates which are the next possible hops of a message with destination d that is currently in q. Possibly, a delivery queue d may not be reachable from a given non-delivery, non-injection queue q. In such a case, $\mathcal{R}(q,d)$ should be equal to $\emptyset$.

The *queue dependency graph* (QDG) corresponding to a set of queues Q and a routing function $\mathcal{R}$ is a directed graph such that its set of vertices is Q and there exists an edge from q_i to q_j $(q_i, q_j \in Q)$ if and only if there exist an injection queue s and a delivery queue d such that $\mathcal{R}$ builds a route from s to d passing through both q_i and q_j, and $q_j \in \mathcal{R}(q_i, d)$. (This definition is related to the one presented in [1] regarding *virtual channels*.) Clearly, if the QDG corresponding to a set of queues and a routing function is acyclic (i.e., it is a DAG), then, the greedy routing algorithm resulting from $\mathcal{R}$ is deadlock free.

Let $D = (Q, A_s)$ be an (acyclic) queue dependency graph. Then, Q is the set of queues and A_s the set of links between the queues. Every nondelivery queue has finite (independent of the size of the network) size. The delivery queues of D will have infinite size, to model the fact that messages are eventually consumed at them. If $q \neq \emptyset$, $Head(q)$ is the first message in FIFO order of q. q can be modified either by deleting its head $(RemoveHead(q))$ or by adding an element m to it as its last element $(Insert(m,q))$. The *Full* function tests if a queue is full. $Level(q)$ is the length of the longest path between any member of $InjectQ$ and q. For every q, $Level(q)$ is finite because D is acyclic.

In previous work, routing functions are built such that the resulting QDG's are acyclic. Although this condition is sufficient to guarantee deadlock freedom, it is too strong, and can be relaxed: the queue dependency graph has to be *dynamically* acyclic, i.e., cyclic wait must not arise in a dynamic environment [24].

This paper uses a model for such dynamically acyclic queue dependency graphs in the generation of practical routing algorithms for hypercubes, meshes, and shuffle-exchanges.

In order to present the results of Sections III, IV, and V some more definitions and terminology will be required. Let $A_d \subset Q \times Q$ be a set such that $A_s \cap A_d = \emptyset$, and, if $(q_1, q_2) \in A_d$, then q_2 is at most one hop away from q_1 in the network. Although it is not necessary, it will be required that if $(q, q') \in A_d$ then $Level(q) \geq Level(q')$. This is not a restriction because if $Level(q) < Level(q')$ then (q, q') can be included in A_s, and D will still be acyclic. Now, let $\tilde{D} = (V, A_s \cup A_d)$ be the extension of D by A_d. Sometimes, D will be called the underlying DAG of $\tilde{D}$. Note that $\tilde{D}$ is not necessarily a DAG. In the following, A_s will be called the *static transition set* and A_d will be called the *dynamic transition set*. Let $\tilde{\mathcal{R}}$ be a routing function on $\tilde{D}$, observing the following conditions: $\forall q, q' \in Q, d \in DelivQ$.

1) $\mathcal{R}(q,d) \subseteq \tilde{\mathcal{R}}(q,d)$.
2) If $q' \in \tilde{\mathcal{R}}(q,d)$ and $q' \notin \mathcal{R}(q,d)$ then $\mathcal{R}(q',d) \neq \emptyset$. This means that if a message can be routed along a dynamic transition, it will still have the possibility of taking a static transition as a next step towards its destination. Therefore, at any moment, every message has a static-transition path that takes it to its destination. In other words, every message will be able to progress towards its target queue through the underlying DAG.
3) $\tilde{D}$ is the QDG corresponding to Q and $\tilde{\mathcal{R}}$.

The Routing Algorithm: Let $\tilde{\mathcal{R}}$ be a routing function and $\tilde{D}$ be the QDG associated with it. Furthermore, suppose that D is the underlying DAG of $\tilde{D}$. The following greedy algorithm can be used to route messages over $\tilde{D}$ from the injection to the delivery queues.

```
Route(q)
/* q is the queue executing the algorithm */
(01)  select q' ∈ {q'' ∈ Q : (q, q'') ∈ A_s ∪ A_d} :
        (not Full(q') and q' ∈ R̃ (q, Dest(Head(q))))
(02)  Insert(Head(q), q')
(03)  RemoveHead(q)
```

It is supposed that once a q finds and selects some q' satisfying the condition in line (01) it gains the access to a place in q', and can execute lines (02) and (03) of the algorithm above. Note that `select` may return a q' satisfying condition in line (01) according to any criterion, as long as it does so if the set of queues satisfying (01) is not empty.

Theorem 2.1: The routing scheme proposed above over $\tilde{D}$, with the extended routing function $\tilde{\mathcal{R}}$, is deadlock free.

The proof of the deadlock freedom of this algorithm is easy. For completeness, it is included in the Appendix.

III. Hypercube Algorithm

In this Section, a fully-adaptive minimal routing algorithm for the hypercube will be presented. A routing function will be built that uses dynamic transitions. So, the QDG associated with this routing function will have cycles. As said above, this routing function should be regarded as an extension of an acyclic routing function (i.e., a routing function whose QDG is acyclic) so as to guarantee that the routing algorithm is deadlock-free. Next, this *underlying* routing function, and how to extend it to achieve the final one will be described.

The routing function that results from routing over the hypercube as hung from node $0\cdots0$ will be used as the underlying acyclic function. This routing algorithm has been presented in [22], for implementing virtual barriers on the hypercube. A similar idea has been used in [21] for implementing a minimal adaptive routing algorithm on the hypercube. The idea on which this *hanging* algorithm is based is the following. Each message is routed in two phases: In phase A, it travels as moving downwards through the network, always moving towards its destination, as much as possible. So, in this phase, each message starts heading to node $1\cdots1$ (which happens to be the node that is opposite to node $0\cdots0$). So, in phase A, each message turns the incorrect 0s in the address of its source node into 1's.

In phase B, every message arrives at its destination by following an upwards path. In this phase, messages move towards node $0\cdots0$. So, in this phase, each message turns the incorrect 1's of its source address into 0's. Therefore, all the required corrections are terminated at the end of this phase. Consequently, each message arrives at its destination.

The following implementation of this algorithm is such that the corresponding QDG is acyclic. Each node n should have two queues, $q_{A,n}$ (associated with phase A), and $q_{B,n}$ (associated with phase B), as well as an injection queue i_n and a delivery queue d_n, as discussed above. During the first phase, messages move through the q_A queues of the nodes they visit. When a message switches phases, it has to start moving through the q_B queues of the nodes visited. The QDG resulting from this implementation is acyclic. Therefore, the algorithm associated with it is deadlock-free.

Fig. 1 shows the QDG of a three-dimensional hypercube. The queues are labeled by the address of the hypercube node where they reside. The upper part of the Fig. shows the queues $q_{A,n}$ linked by solid arrows representing the possible transitions a message can take from one node to a neighboring node of the hypercube while changing incorrect 0's into 1's. The lower part of the figure shows a similar arrangement for the $q_{B,n}$ queues. The solid arrows represent node transitions a message can take while changing incorrect 1's into 0's. Finally, the dashed arrows show the change of phases in the routing. Notice that these arrows link queues residing in the *same* hypercube nodes, and thus these transitions do *not* involve communication.

As messages are forced to correct first the incorrect 0's into 1's and only afterwards the incorrect 1's into 0's, congestion around node $1\cdots1$ takes place. This technique has been modified to yield a more flexible algorithm in [21] but deadlock freedom was ensured at the expense of eliminating possible paths from the network and thus, the method yields a partially-adaptive router. Furthermore, the algorithm in [21] needs 3 queues per node.

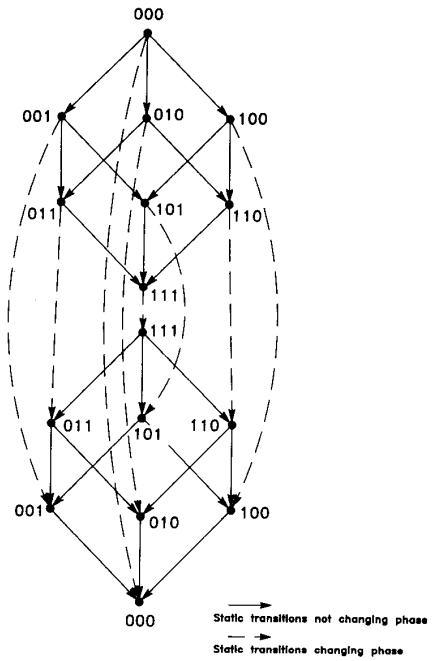


Fig. 1. A 3-hypercube hung from node 000.

Now, a fully adaptive deadlock-free algorithm based on only two queues will be shown. Dynamic transitions will be added to the QDG in such a way that messages are allowed to change incorrect 1's into 0's while being in phase A if the message finds place in the q_A queue of the corresponding node, at a certain moment. The resulting algorithm is as follows. Each message is injected, and starts moving through the q_A queues of the different nodes it visits (phase A) *while it has any 0 to correct into 1*. During this phase (moving through q_A queues), each message can correct *any* of the incorrect dimensions. After performing the last 0 to 1 correction, the message must enter the q_B queue of the corresponding node, and will start making the 1-to-0 corrections needed until it arrives at its destination node.

In Fig. 2, the dynamic transitions have been added to those shown in Fig. 1. The new transitions, drawn as thick arrows, connect q_A queues in the upper part of the figure. The graph linking queues q_A has become a hypercube as all transitions between neighboring nodes are possible by using these queues. A message will travel adaptively along a minimal path in this part of the graph (visiting queues q_A) as far as it has at least one 0-to-1 transition to make, i.e., the message still needs to traverse a solid thin arrow. Once all 0-to-1 transitions are exhausted, the message *must* change phases (represented by the dashed-arrow transition to q_B queues) and traverse adaptively in the lower part of the Fig. (i.e., by using q_B queues).

As is seen, a message may take on any available path to progress toward its destination and never revisits any node.

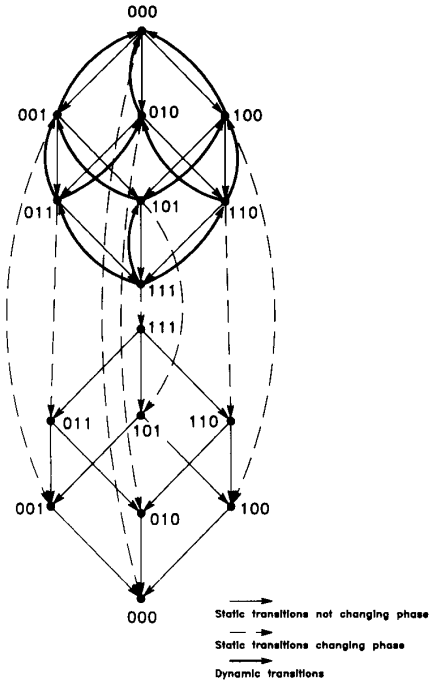


Fig. 2. A 3-hypercube hung from node 000 with dynamic transitions.

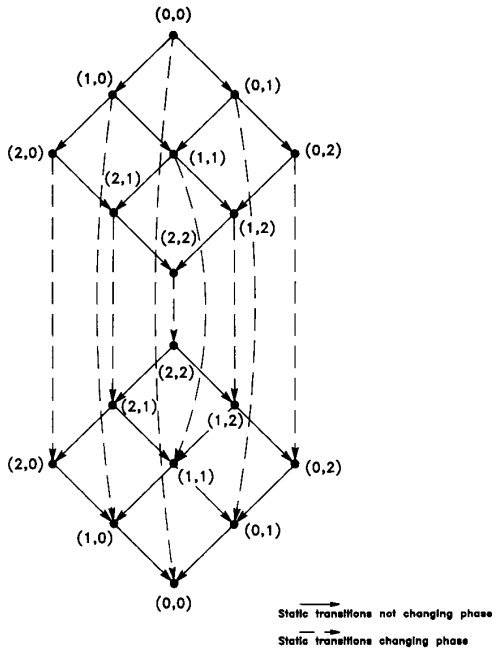


Fig. 3. A 3-mesh hung from node $(0,0)$.

Furthermore, the travelled number of hops is exactly the Hamming distance between the source and the destination nodes, and thus, the algorithm is minimal. The adaptivity of the above algorithm is materialized by having an adequate flow control. This flow control depends on other features of the node model as will be discussed in Section VI.

Thus, with the queue policy just outlined, the resulting routing algorithm is deadlock-free, and allows each message to wait to correct any of the possible dimensions it has to correct. This algorithm requires only two queues per node, plus the injection and delivery queues, and is fully-adaptive minimal. The formal definition of the routing function is given in the Appendix.

It is worth remarking that no particular congestion should be expected near node $1\cdots1$ as messages are allowed to move upwards even if they are in phase A, as a result of the newly added dynamic transitions. In fact, this algorithm is the only candidate known for the hypercube that might exhibit deterministic worst-case time performance $O(\log N)$ for static injection (also called batch routing). As this conjecture has not been proved or disproved[1], performance has been tested experimentally under different traffic conditions, structured communication patterns, and models of injection. Simulation results of this algorithm for hypercubes of up to 16-K nodes are reported in Section VII. In all cases, it was demonstrated that *no congestion arises in any node of the network* (see Section VII), showing the advantage of the new algorithm over the conventional oblivious e-cube technique.

[1] Actually, the complexity of the theoretical answer may be formidable.

IV. Mesh Algorithm

A routing algorithm for the mesh will be presented here in terms of the ideas of dynamic transitions. The scheme is minimal and deadlock free. Although the following description focuses on two-dimensional meshes, the technique can be easily generalized for k-dimensional meshes, for any arbitrary k.

The key idea is similar to the one presented for the hypercube network. First, a partially adaptive minimal algorithm having a QDG that forms an acyclic graph will be shown. Second, this technique will be extended to a fully-adaptive routing by using dynamic transitions.

The partially adaptive algorithm is based on the idea of "hanging" the mesh from the $(0,0)$ and $(n-1,n-1)$ nodes and consists of two phases. In phase A the messages move toward their destination by visiting nodes in such a way that if a message passes from (x,y) to (x',y') in one routing step, then $x < x'$ or $y < y'$. In phase B, messages visit nodes with lower number instead of those with higher number. In other words, the mesh is hung from node $(0,0)$ in phase A and the messages visit nodes with higher level, where the level of (x,y) is $x+y$. In phase B, the mesh is hung from node $(n-1,n-1)$ and the nodes are visited in decreasing level order. Each message starts the routing process in phase A. After all of the steps that could be taken in phase A have been completed, the message enters phase B. Certain messages arrive at their destinations by taking steps only in phase A (resp. phase B). This scheme can be implemented using two queues in each node, q_A for phase A messages and q_B for phase B messages.

In Fig. 3, a diagram of the QDG for a 3×3 mesh is shown. The explanation of this figure is completely analogous to that of the hypercube case shown in Fig. 1. The routing scheme is

deadlock free, because the queue dependency graph is acyclic. Also, this scheme has some degree of adaptivity, because messages can use all the descending (resp. ascending) paths towards their destinations in the first (resp. second) phase. However, many pairs of communicating nodes can be served by a unique path as, for example, is the case of a message travelling from node $(0,2)$ to node $(2,0)$.

Formally, the routing function[2] is given by the following:

$$\mathcal{R}(i_{(x,y)}, d_{(z,w)}) = \begin{cases} q_{A,(x,y)}, & \text{if} \quad z > x \text{ or } w > y, \\ q_{B,(x,y)}, & \text{if} \quad z \leq x \text{ and } w \leq y, \end{cases}$$

$$\mathcal{R}(q_{A,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)}, & \text{if} \quad x = z \text{ and } y = w, \\ q_{A,(x+1,y)}, & \text{if} \quad z > x, \\ q_{A,(x,y+1)}, & \text{if} \quad w > y, \\ q_{B,(x,y)}, & \text{if} \quad z \leq x \text{ and } w \leq y, \end{cases}$$

$$\mathcal{R}(q_{B,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)}, & \text{if} \quad x = z \text{ and } y = w, \\ q_{B,(x,y-1)}, & \text{if} \quad w < y, \\ q_{B,(x-1,y)}, & \text{if} \quad z < x. \end{cases}$$

This routing function allows some degree of adaptivity. But suppose that some message starts from node (x,y) towards its destination (v,w), and let $v < x$ and $w > y$. Following the function above, this message has only one path, namely correct its second dimension, change phase and correct its first dimension. So, it has no adaptivity at all.

In the following, this scheme will be extended to a fully adaptive one, that is still deadlock-free and uses the same number of queues. This is done by allowing messages that have not finished their phase A to take phase-B steps (but still visiting q_A queues). These steps are dynamic transitions, using the terminology of Section II. The phase change mechanism is the same as in the previous scheme. In phase B, the messages still have to go through ascending paths. The resulting algorithm is such that every message *always* has the chance of taking a static transition, as messages keep visiting q_A queues while taking dynamic transitions.

Fig. 4 shows the dynamic transitions among the q_A queues drawn as solid thick arrows. The explanation of this figure is identical to that of Fig. 2 shown in Section III.

Formally, the routing function of the fully-adaptive mesh routing algorithm is given as follows.

$$\tilde{\mathcal{R}}(i_{(x,y)}, d_{(z,w)}) = \begin{cases} q_{A,(x,y)}, & \text{if} \quad z > x \text{ or } w > y, \\ q_{B,(x,y)}, & \text{if} \quad z \leq x \text{ and } w \leq y, \end{cases}$$

$$\tilde{\mathcal{R}}(q_{A,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)}, & \text{if} \quad x = z \text{ and } y = w, \\ q_{A,(x+1,y)}, & \text{if} \quad z > x, \\ q_{A,(x-1,y)}, & \text{if} \quad z < x, \text{ and } w > y \\ q_{A,(x,y+1)}, & \text{if} \quad w > y, \\ q_{A,(x,y-1)}, & \text{if} \quad z > x \text{ and } w < y, \\ q_{B,(x,y)}, & \text{if} \quad z \leq x \text{ and } w \leq y, \end{cases}$$

$$\tilde{\mathcal{R}}(q_{B,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)}, & \text{if} \quad x = z \text{ and } y = w, \\ q_{B,(x,y-1)}, & \text{if} \quad w < y, \\ q_{B,(x-1,y)}, & \text{if} \quad z < x. \end{cases}$$

Using Theorem 2.1 and the fact that the routing function was built using the methodology of Section II it can be seen that the algorithm is deadlock-free. In addition, it can be easily verified that the routing function is fully-adaptive minimal, and so, livelock cannot take place.

[2] In this section, $\mathcal{R}(a,b)$ is the set of all the right members satisfying the associated condition involving a and b. The same applies to the definition of $\tilde{\mathcal{R}}$ below.

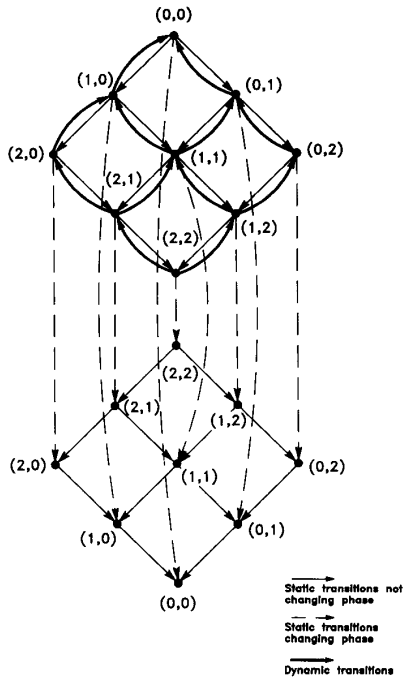


Fig. 4. A 3-mesh hung from node $(0,0)$ with dynamic transitions.

A fully-adaptive and minimal routing technique for packet-switching over tori can be achieved using four queues per node (plus an injection and delivery queue per node) following an idea similar to the one presented in [34] for worm-hole routing over tori [39].

V. Shuffle-Exchange Algorithm

Although the shuffle-exchange network is not a practical interconnection, finding deadlock-free routers has attracted some attention [1]. Previous algorithms for deadlock-free routing are oblivious and use a logarithmic number of queues (see [1]). The algorithm presented in this section is the first known in the literature that requires a constant number of queues and provides some degree of adaptivity. Furthermore, the algorithm is this section has a worst case of $3 \log N$ routing hops. This technique requires only 4 queues per node for its implementation.

First, consider a 2^n-node shuffle-exchange network as without the exchange links. Each connected component of the graph will be called a *shuffle cycle*. Note that every node in a shuffle cycle has the same number of 1's in its binary address. Then, the *level* of a shuffle cycle can be defined as the number of ones in the address of any of its nodes. The idea of the algorithm is to break the shuffle cycles using the technique presented in [1] and then, visit the cycles so as to avoid deadlock.

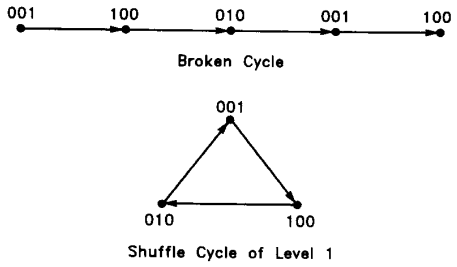


Fig. 5. Breaking a cycle in a 3-shuffle-exchange.

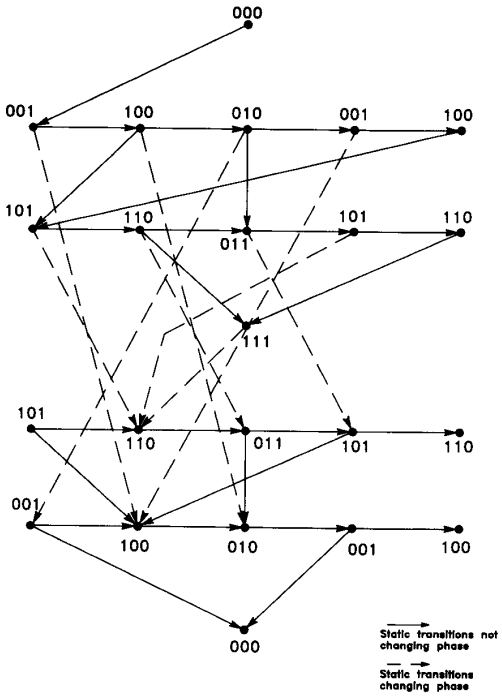


Fig. 6. A 3-shuffle-exchange hung from node 000.

The routing strategy can be defined in two phases. In the first one, messages can move from one shuffle cycle to another whenever the new cycle has higher level. In the second phase, messages visit the shuffle cycles in decreasing order with respect to their level. The routing algorithm consists of visiting the dimensions of the address to correct twice, once in each phase. In each phase, dimensions are visited using the shuffle links. Consequently, every path has at most $3n$ steps: at most $2n$ shuffle steps and at most n exchange steps (see Fig. 6).

After going through a shuffle link, every message has to know which dimension of the destination corresponds to the current least significant bit so as to know whether the least significant bit has to be corrected or not. So, each message must record the number of shuffle links it has already traversed. This is necessary to compare the least significant

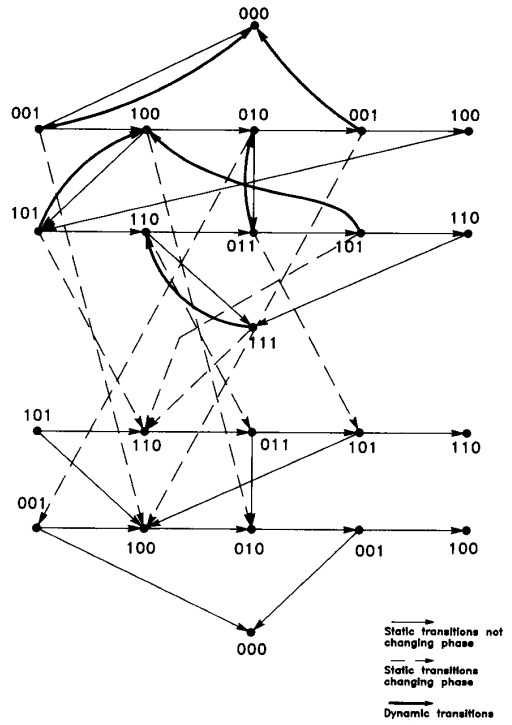


Fig. 7. A 3-shuffle-exchange hung from node 000 with dynamic transitions.

bit of the current node address with the corresponding bit of the destination address so as to decide what to do as the next step. If these bits disagree, that dimension will have to be corrected at that step or not depending on the phase the message is in. In the first phase, a dimension will be corrected if it has to be changed from 0 to 1. Note that this restriction implies that the new cycle has higher level. In the second phase, the reverse direction of the exchange links is used. Only will a change from 1 to 0 be allowed. So, the level of the cycles that are visited decreases during the second phase.

The routing function described above can be implemented using four queues per node, as it is necessary to break the shuffle cycles twice: once for each phase, and two queues are needed to break a shuffle cycle once.

Next, the modification of the routing by adding dynamic transitions is presented. Basically, the main change introduced is that a message will be allowed to traverse an exchange link that corrects the current dimension from 1 to 0 even if the message is in its first phase. In other words, a message will be allowed to correct a 1 to 0 if it happens to find place to do it during the first phase. If not, that dimension will have to be changed during the second phase. As a result of these changes, the resulting routing algorithm is adaptive, as a given message may take alternative paths as a consequence of local congestion: e.g., it may or may not correct a 1 into a 0 during the first phase (see Fig. 7). A formal description of this routing function can be found in [40].

VI. The Model of the Routing Node

In this section, a possible node model will be presented. In Section VI-A, a detailed description of a node to implement the algorithm presented in Section III for the hypercube will be given. Sections VI-B and VI-C will be devoted to the nodes for implementing the algorithms presented in Section IV for the mesh and in Section V for the shuffle-exchange networks, respectively.

As described above, a message can move from a queue to another queue following dynamic or static transitions. If $(q_1, q_2) \in A_s \cup A_d$, then q_2 will receive messages from q_1. So, there must exist a physical connection between q_1 and q_2. If q_1 and q_2 belong to adjacent nodes, then this physical connection is the physical link between the two nodes. On the other hand, if q_1 and q_2 belong to the same node, then, there must exist an internal connection between the two queues so as to allow internal passage of message within the nodes.

A key idea in the control-flow of the nodes is the introduction of buffers that serve the links of the networks. This idea has also been used in many other research works (see, e.g., [21]). Buffers allow to decouple the link activities (i.e., sending and receiving messages) from those of the central queues and switching. In the case of the algorithms presented in this paper, buffers are also important because the conditions that allow messages to take dynamic transitions can be checked locally in each routing node and thus, no direct communication between two queues residing in different nodes will be necessary. As it will be explained below, queues will receive messages from buffers, as well as from other queues in the same node. Thus, given a queue q, some fair policy must be implemented so as to guarantee fair access to q to all the resources that may want to access q.

Each node will have both an injection and a delivery queue, as explained above, as well as all the queues used by the routing algorithm. Each physical link will have associated with it input and output buffers. In general, there will be two types of buffers associated with each physical link: those associated with dynamic transitions and those with static transitions. Consider link j, incident to nodes n and n'. If traffic corresponding to dynamic transitions can enter node n from node n' through link j, then link j will have an input buffer in node n and an output buffer in node n' associated with the dynamic transitions. So, if a message has to go out of node n' through link j via a dynamic transition, it will be placed in the output buffer corresponding to dynamic traffic of link j if this buffer is empty, and it will arrive at the input buffer in node n that is associated with both dynamic transitions and link j. If traffic corresponding to static transitions can enter queue q in node n through link j, then link j will have an input buffer associated with queue q in node n and an output buffer associated with queue q in node n'. So, if some queue q' in node n' wants to send a message through link j to queue q via a static transition, then it will place the message in the output buffer corresponding to link j and queue q in node n'.

Next, this node model will be illustrated on the interconnection networks and algorithms presented above. A more detailed description of the activity of a node during routing can be found in Section VII-A where extensive simulations are reported.

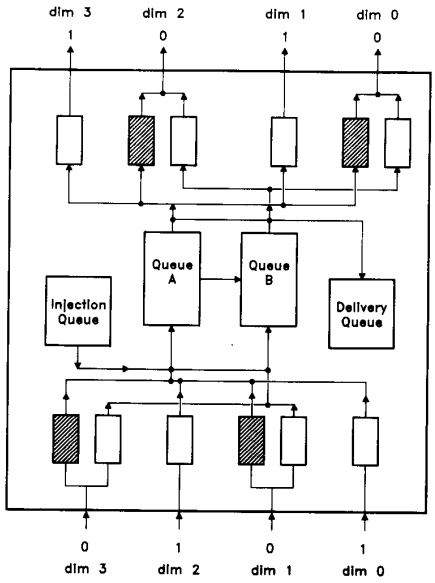


Fig. 8. Node 0101 of the 4-Hypercube.

A. Hypercube Node Design

This section will focus on the hypercube routing technique presented in Section III.

Every node will have an injection and a delivery queue, and two other queues, as described in Section III: queue q_A and queue q_B. As a result of the routing function, messages will move from q_A to q_B of the node in which they switch from phase A to phase B. So, q_A must be connected to q_B in each node. According to the routing function, and the node model given above, each physical link will have either an input and two output buffers, or two input and one output buffers, depending on the number of the corresponding node.

Consider a four-dimensional hypercube. Furthermore, consider node 0101 (see Fig. 8). Messages arriving through the link corresponding to dimension 3 are messages that are changing that dimension from 1 to 0. So, these messages are either in phase B, and so, they will enter queue q_B through a static transition, or in phase A, and they will enter queue q_A through a dynamic transition. Therefore, this link will have two input buffers: one associated with queue q_B (static transitions), and one associated with queue q_A (dynamic transitions).

Regarding the output buffers associated with the link corresponding to dimension 3, notice that messages going out of node 0101 through that link turn the 0 in the most significant dimension into a 1. So, this link will only be taken by messages in phase A (static transitions), i.e., by messages at queue q_A. Therefore, this link will only have one output buffer, and it will be associated with queue q_A (static transitions).

Regarding the link corresponding to dimension 2, this link will be taken by messages changing a 0 into 1 in that

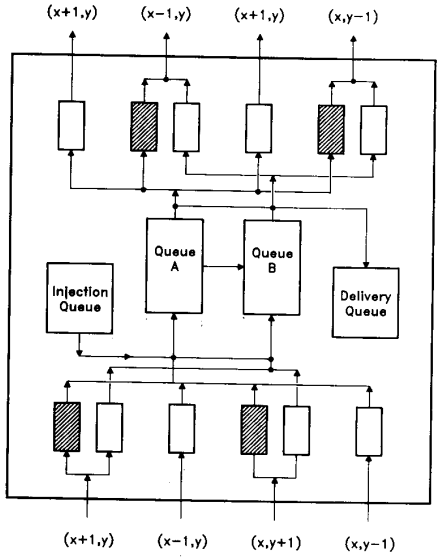


Fig. 9. The node for the Mesh.

dimension. So, this physical link will have one input buffer and two output buffers, following a reasoning similar to the one just above.

Each physical link will thus have associated three buffers: one input buffer, one output buffer and the third one will be either an input or an output one, depending on the address of the corresponding node. The reader may assume that a practical design would incorporate the same number of input and output buffers per physical link making all nodes in the network identical.

B. Mesh Node Design

In this section, the implementation of the routing function presented in Section IV will be presented. Only three buffers per physical link, and two queues, q_A and q_B, per node are needed. As before, each node must have an injection and a delivery queue.

Consider a generic node (x, y) of a $(k \times k)$-mesh where $0 < x, y < k - 1$ (see Fig. 9).[3] Consider links $((x, y), (x - 1, y))$, and $((x, y), (x, y - 1))$. Each of these links has only one associated input buffer in node (x, y), because (x, y) can only receive messages in their first phase, and taking static transitions, through these links. Therefore, these messages can only enter queue q_A. On the other hand, each of these links has two associated output buffers in (x, y). One of these buffers holds those messages that are to be sent through a dynamic transition during the first phase. So, this buffer receives messages only from q_A. The other buffer holds those messages that take static transitions during the second phase. So, this buffer receives messages only from q_B.

Now, consider links $((x, y), (x + 1, y))$ and $((x, y), (x, y + 1))$. Each of these links has associated with it only one output buffer in (x, y), connected to q_A, that holds messages using static transitions during the first phase. On the other hand, each of these links has two associated input buffers in (x, y). One of these buffers is connected to q_A, and it is used by messages taking dynamic transitions during their first phase. The other one is connected to q_B, and holds those messages that take static transitions during the second phase. As each buffer is connected to only one queue, q_A has to be connected to q_B in order to allow the messages to switch from phase A to phase B.

[3] For those nodes which are on an "edge" of the mesh, their design is identical but some buffers are never used.

C. Shuffle-Exchange Node Design

This section will focus on how to implement the algorithm presented in Section V for the shuffle exchange network. The node model that will be described varies from node to node, depending on whether the number of the node is odd or even.

Each node has four queues, plus the injection and delivery queues. Each of these queues has an input buffer and an output buffer associated with the shuffle link. Queues q_1 and q_2 and their corresponding buffers are used during the first phase of the algorithm. The other two queues, and their corresponding buffers are used during the second phase of the algorithm. In addition, each node has three buffers associated with the exchange link. If the node number is odd, there is one input buffer associated with the exchange link, and it is connected to queues q_1 and q_2, because messages arriving from this link are changing the least significant bit from 0 to 1 (the node is odd), and so, these messages are taking a static transition during phase A. There are two output buffers associated with the exchange link. One of them is used by phase A messages taking a dynamic transition, and the other one is used by phase B messages taking a static transition. If the node is even, there are two input buffers associated with the exchange link: one corresponding to phase A messages taking a dynamic transition, connected to queues q_1 and q_2, and the other one, corresponding to phase B messages taking a static transition, connected to queues q_3 and q_4. This is so because messages arriving from this link are changing the least significant bit from 1 to 0 (the node is even). There is only one output buffer associated with the exchange link in this case. This buffer is used by phase A messages taking a static transition.

Fig. 10 shows a generic node model that allows the implementation of both the even and the odd case.

VII. Simulation Results

Mathematical analyses yielding bounds for the behavior of routing techniques have received a great deal of attention. Oblivious routing schemes have been analyzed for several networks and probabilistic bounds on the performance of different algorithms with static injection have been shown [41], [6], [42], [43], [2], [3], [26]. Even when mathematical bounds can be estimated, the practical performance of routing algorithms is important. To this end, simulation results for static and dynamic injection models have been reported by several authors [41], [14], [44], [45].

On the other hand, deterministic bounds have been proved for sorting-based algorithms [46], [47] and for adaptive algorithms based on the multibutterfly construction [4], [5]. In

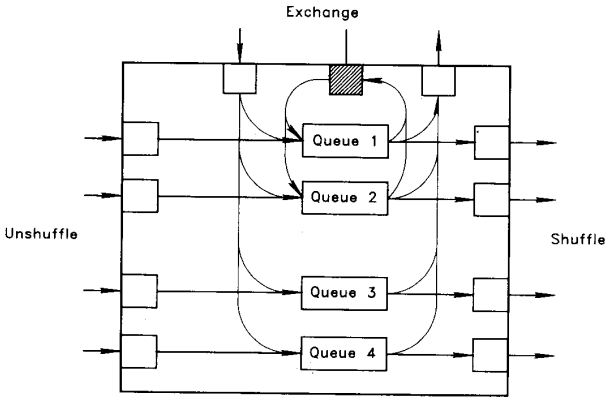


Fig. 10. The generic node for the Shuffle-Exchange.

general, the problem of finding mathematical bounds on the performance of adaptive routing for general networks seems a challenging problem.

For dynamic message injection, several attempts to model and measure the performance of interconnection networks are known [13], [48]–[50]. Dynamic injection presents a paramount complexity to the mathematical analysis of routing techniques and in many cases, simulations of the involved networks have been the only alternative to show the performance achieved under different injection loads, topology of interconnection, queue size, and arbitration of conflicting resources.

The main goal of this Section is to report a number of simulation results on the routing techniques for the hypercube and the mesh presented in Sections III and IV. The simulation work is aimed at showing the practicality of the new routing ideas for different communication patterns and models of injection. Also, some comparisons are made with other routing algorithms.

According to the node models described in Sections VI-A and VI-B, each node will have both an injection and a delivery queue, as well as the two queues used by the routing algorithm. The injection queue will have size 1, modeling the fact that each node can generate at most one message in each cycle and that no node will be allowed to inject a new message until all the previously injected messages have begun their route. As will be described later, the delivery queue is not needed. The two central queues will have the (arbitrary) size of 5 messages each. So, the queue size will be independent of the network size.

Throughout the rest of this paper, the fully-adaptive minimal routing techniques described in Sections III and IV will be referred to as *FULL*. The routing algorithms that result from removing all the dynamic transitions from *FULL* will be called *ADAPT*. These routing algorithms are still adaptive and minimal but not fully-adaptive as some possible minimal paths are excluded. The routing algorithm that results from choosing one only among the static transitions in a deterministic way will be called *OBLIVIOUS*. This last technique has no adaptivity at all.

Sections VII-B and VII-C present the simulation results for the hypercube and the mesh, respectively.

A. Network Activity

This Section describes the activity of the network used in the simulations reported in Sections VII-B and VII-C. Because latency will be measured in terms of *routing cycles*, a definition of the amount of work involved in a cycle is needed.

Every routing cycle is divided into two parts, a *node* cycle and a *link* cycle. During the node cycle, the queues send their messages to a suitable output buffer. The queues are scanned in FIFO order, and all the messages that find a useful output buffer empty are allowed to move there. Next, each node scans all of its input buffers and its injection buffer and moves the messages to the corresponding queue, if possible. During this scanning, messages addressed to that node are consumed and removed from the network. In order to avoid starvation, the input and injection buffers of a node are arranged in a cycle, and they are scanned cyclically, beginning with the first buffer that has not found an empty slot in a central queue for its message in the previous cycle. Once this step has been performed, the node is allowed to inject a message in its injection buffer, provided that this buffer is empty and the node has a message to inject. This step ends the node cycle. Note that every message needs at least *two* routing cycles to pass through a node.

In the link cycle, each link[4] with a message in its associated output buffer sends it to the corresponding input buffer, provided that this buffer is empty. Note that some links have two output buffers associated, but only one message can pass through the link in a single cycle. So, it is necessary to manage the link in a fair way, to avoid that any message starves in an output buffer.

Injection Model: There are two injection models: the *static* injection model, in which every node has a fixed number of messages to inject, and the *dynamic* injection model, in which each node wants to inject at arbitrary moments. In the first case, the routing process begins when each node injects its first message, and ends when the last message arrives at its destination. Some interesting parameters to measure for this injection model are $L_{\max}$, the maximum latency, and L_{avg}, the average latency.

The dynamic injection model presents a number of phenomena that do not appear in the static case. Dynamic injection leads to infinite processes. So, in order to observe the interesting phenomena, the routing process has to be truncated at some point. Another problem is that if the average number of packets that a node tries to inject per cycle, λ, is too large, the system can become *saturated*, i.e., $L_{\max}$ grows without bound. So, the maximum λ that maintains the system unsaturated is a key parameter to measure.

In addition, in the dynamic model it is important to measure the network throughput τ, which is defined as the average number of messages that are injected in a cycle per node. Some simple bounds are known for τ. If N is the number of nodes of the network, B the bisection of the network [26] [51] and c the proportion of messages that cross the bisection,

[4]Here, each bidirectional link is considered as two links, one in each direction.

then

$$\frac{N\tau c}{2} < B. \tag{1}$$

This formula means that, in every cycle, the number of messages injected in the network in that cycle that are going to cross the bisection should be less than the bisection in order to avoid saturation. So

$$\tau < \frac{2B}{Nc} = \tau_{\max}. \tag{2}$$

This bound is used to express the obtained throughput as a percentage of this theoretical maximum. The parameter λ will be expressed in the same way.

Communication Patterns and Traffic Characteristics: The performance of networks should be measured for different patterns of communication and traffic characteristics. Two communication patterns will be used.

- *Random Routing:* Every message chooses its destination randomly. This models the unstructured pattern of communication that is present in many applications.
- *Fixed Permutations:* In this case, a permutation σ is fixed in advance. A node p injects all of its messages with destination $\sigma(p)$. In the static injection model, this pattern is useful when a large structure is embedded in a smaller network (e.g., an $m \times m$ matrix in an $n \times n$ mesh, $m \gg n$). Dynamic injection mimics the same kind of pattern when the number of messages per node is much greater than the number of nodes, so the injection can be thought of as continuous. In particular, the following permutations were simulated.

  1) *Transpose:* In the mesh, node (x, y) will send messages to node (y, x). In the hypercube, the binary address of the node will be split into halves, and these halves will be swapped (if the dimension of the hypercube is odd, then the middle bit will remain unchanged).
  2) *Complement:* In the hypercube, the complement is defined by complementing all the bits in the binary address of the node.
  3) *Leveled Permutation:* As defined in Sections III and IV, each node has an associated level. So, a leveled permutation is a random permutation in which every node sends messages to some node in the same level.
  4) *Bit Reversal:* In the hypercube, this permutation is obtained by reversing the bit-string of the binary address of the node and in the $n \times n$-mesh, by concatenating the $\lceil \log n \rceil$ bits of each coordinate and then reversing the resulting string.

  Some of these permutations are known to create congestion in the hypercube or in the mesh if oblivious routing algorithms are used. For example, the e-cube algorithm suffers from severe congestion for traffic patterns such as the *Tranpose*. Bit reversal is known to produce congestions for the mesh and torus [25] for the $x - y$ algorithm.

TABLE I
RANDOM ROUTING, 1 PACKET

n	N	L_{avg}	$L_{\max}$
7	128	8.28	13
8	256	9.37	15
9	512	9.94	17
10	1024	10.96	19
11	2048	12.09	21
12	4096	13.08	25
13	8192	14.03	27
14	16384	15.04	29

TABLE II
COMPLEMENT, 1 PACKET

n	N	L_{avg}	$L_{\max}$
7	128	15	15
8	256	17	17
9	512	19	19
10	1024	21	21
11	2048	23	23
12	4096	25	25
13	8192	27	27
14	16384	29	29

TABLE III
TRANSPOSE, 1 PACKET

n	N	L_{avg}	$L_{\max}$
7	128	7.03	13
8	256	9.03	17
9	512	9.03	17
10	1024	11.09	21
11	2048	11.09	21
12	4096	13.13	25
13	8192	13.13	25
14	16384	15.23	29

TABLE IV
LEVELED PERMUTATION, 1 PACKET

n	N	L_{avg}	$L_{\max}$
7	128	7.43	13
8	256	8.04	13
9	512	9.24	17
10	1024	10.10	21
11	2048	10.98	21
12	4096	12.06	25
13	8192	13.07	25
14	16384	14.03	29

B. The Simulations for the Hypercube

In this section, simulation results for the routing algorithm proposed in Section III will be shown for the hypercube. Recalling (2), for the n-dimensional hypercube, $B = 2^n/2 = N/2$. So,

$$\tau_{\max} = \frac{1}{c}.$$

This means that $\tau_{\max}$ does not depend on the size of the network. Note that since c is a proportionality factor, $c \leq 1$,

TABLE V
RANDOM ROUTING, n PACKETS

n	N	L_{avg}	L_{max}
7	128	8.10	15
8	256	9.25	17
9	512	10.29	20
10	1024	11.33	22
11	2048	12.52	25
12	4096	13.76	27
13	8192	15.02	30
14	16384	16.54	32

TABLE VI
COMPLEMENT, n PACKETS

n	N	L_{avg}	L_{max}
7	128	15.00	15
8	256	17.00	17
9	512	19.00	19
10	1024	21.00	21
11	2048	24.99	30
12	4096	28.61	35
13	8192	32.74	39
14	16384	36.23	44

TABLE VII
TRANSPOSE, n PACKETS

n	N	L_{avg}	L_{max}
7	128	7.07	15
8	256	9.23	19
9	512	9.13	21
10	1024	12.27	26
11	2048	12.40	32
12	4096	16.01	37
13	8192	16.22	36
14	16384	20.49	43

TABLE VIII
LEVELED PERMUTATION, n PACKETS

n	N	L_{avg}	L_{max}
7	128	7.77	15
8	256	8.52	17
9	512	9.77	19
10	1024	10.78	23
11	2048	11.77	25
12	4096	13.17	28
13	8192	14.60	32
14	16384	16.03	37

TABLE IX
RANDOM ROUTING, $\lambda = 1$

n	N	L_{avg}	L_{max}	τ
7	128	8.57	21	98
8	256	9.67	23	97
9	512	10.82	26	96
10	1024	12.10	30	93
11	2048	13.47	35	89
12	4096	15.01	37	85
13	8192	16.58	44	81
14	16384	18.30	49	76

TABLE X
COMPLEMENT, $\lambda = 1$

n	N	L_{avg}	L_{max}	τ
7	128	24.11	33	75
8	256	24.11	37	68
9	512	28.49	43	61
10	1024	33.32	52	55
11	2048	39.29	58	49
12	4096	45.60	68	45
13	8192	52.87	79	41
14	16384	60.70	90	38

TABLE XI
TRANSPOSE, $\lambda = 1$

n	N	L_{avg}	L_{max}	τ
7	128	7.02	14	100
8	256	10.33	25	94
9	512	10.33	25	94
10	1024	14.67	36	83
11	2048	14.67	36	83
12	4096	15.78	49	73
13	8192	20.31	54	71
14	16384	27.33	66	61

TABLE XII
LEVELED PERMUTATION, $\lambda = 1$

n	N	L_{avg}	L_{max}	τ
7	128	9.11	32	97
8	256	9.75	34	97
9	512	11.28	37	94
10	1024	12.47	43	91
11	2048	13.50	48	89
12	4096	15.17	56	84
13	8192	16.91	53	80
14	16384	18.46	57	75

and then $\tau_{max} \geq 1$. Also, because of the node model, at most one packet can be injected in each cycle. So, for the dynamic injection case, λ was fixed to 1.

In the tables, N is the number of nodes of the hypercube, n is the number of dimensions of the hypercube. Also, L_{avg} is the average latency of the messages, and L_{max} is the maximum latency any packet experienced, as before.

Tables I–IV show the results for static injection when each node injects one packet. Tables V–VIII show the results for static injection when each node injects n packets. This number of packets has been tried because there exists a good theoretical bound [6] for this pattern of communication using random routing. These tables show that the fully-adaptive algorithm does not exhibit the typical congestion encountered in oblivious routers such as the e-cube. e-cube is well-known to have congestion for some structured communication patterns such as *Transpose*.

Tables IX–XII show the results for dynamic injection. It should be noted that no saturation arises in the hypercube, but τ degrades. This is due to the fixed size of the queues.

Under all traffic conditions, the algorithm performs with no congestion thus demonstrating the importance of adaptivity in hypercube routing.

C. The Simulations for the Mesh

In this section, the simulation results for the algorithm described in Section IV for the mesh will be presented. In this topology, the algorithms *FULL, ADAPT,* and *OBLIVIOUS* will be compared. It is important to notice that *OBLIVIOUS* is a variation of the e-cube algorithm for the mesh topology. Only the results for dynamic injection will be considered here. Recalling (2), for an $n \times n$-mesh, $B = n, N = n^2$, so $\tau_{\max} = 2/(nc)$. This means that $\tau_{\max}$ decreases very rapidly with n. The simulations focused on only one network size, namely the $32 \times 32 = 1K$-mesh, and tried to show the performance of the algorithms *FULL, ADAPT,* and *OBLIVIOUS* for different λ's.

One modification was made to the *FULL* algorithm presented above for the mesh. Since the links are a scarce resource in the mesh, the excessive use of dynamic transitions could spoil all the routing because a message trying to get through one of these transitions can delay a message trying to make a static transition through the same physical link. Hence, it was decided that a message can be moved to an output buffer associated with a dynamic move only if no message is waiting in the static buffer associated with the same link.

For all the communication patterns simulated, there are plots for $L_{\text{avg}}, L_{\max}$, and τ as a function of λ. λ and τ are expressed as a percentage of $\tau_{\max}$. Simulations were performed for injection rates going from 10% of $\tau_{\max}$ to its 80%, in intervals of 5%. After the network saturates, the values in the plots shown below have no meaning at all: they are finite only because the simulation is stopped.

In the following plots, the *FULL* line stands for the *FULL* algorithm, the more densely dashed line stands for the *ADAPT* algorithm and the less dense one for the *OBLIVIOUS* algorithm.

The results are the following.

1) *Random Routing:* In this case, $c = 1/2$, so $\tau_{\max} = 4/n = 0.125$. Fig. 11 shows that the algorithm *FULL* was able to accept injection up to the 75% of $\tau_{\max}$ before saturation, while both *ADAPT* and *OBLIVIOUS* could only accept up to the 50%. Before saturating, L_{avg} and $L_{\max}$ were slightly better for the *FULL* algorithm than for the other two. Fig. 11 also shows that for *FULL* the maximum τ is reached when $\lambda = 75\%\tau_{\max}$, $\tau = 3/n = 0.09375$.
2) *Transpose:* Here, $c = 1/2$ again, and $\tau_{\max}$ is equal to that of Random Routing. For transposing, *ADAPT* and *OBLIVIOUS* are the same algorithm. That is because in *ADAPT* a message going from (x, y) to (x', y') can use the adaptivity of *ADAPT* only if ($x < x'$ and $y < y'$) or ($x > x'$ and $y > y'$). But when going from (x, y) to (y, x), none of these relations can be true, so there is no adaptivity at all and *ADAPT* should behave equal to *OBLIVIOUS*.

Fig. 12 shows that the three algorithms behave very similarly for λ less than 25% of $\tau_{\max}$. *OBLIVIOUS* and *ADAPT* saturate for any λ higher than 25% of $\tau_{\max}$, while *FULL* tolerates higher loads. The *FULL* algorithm sustains up to 35% of the maximum throughput, ($\tau = 1.48/n = 0.04625$) before the maximum latency begins to grow, while *OBLIVIOUS* (and *ADAPT*) up to 25%, with higher latency than that displayed by *FULL*. Also, maximum latency grows in a smoother way in *FULL* than in *OBLIVIOUS*.

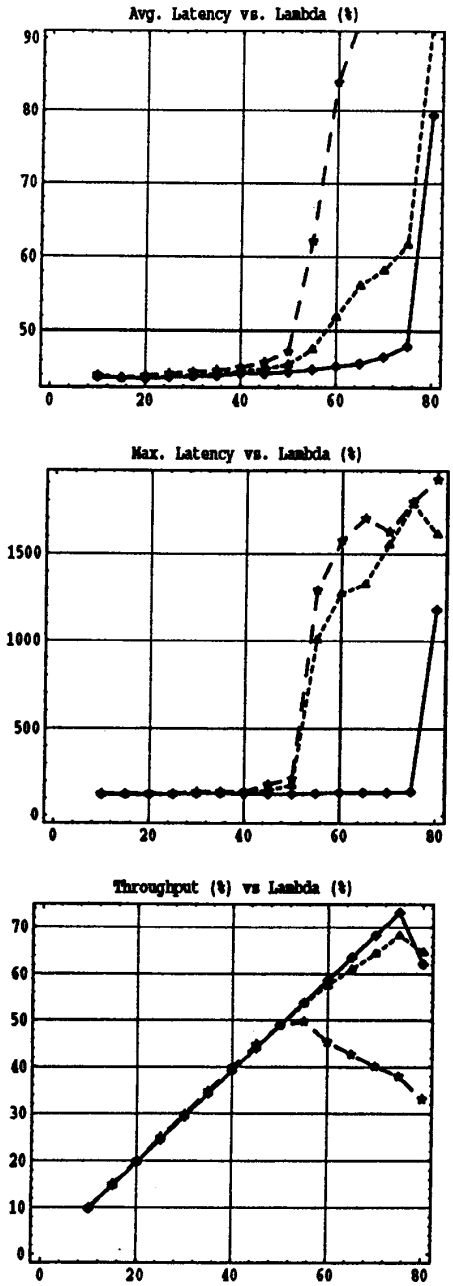


Fig. 11. Results for Random Routing.

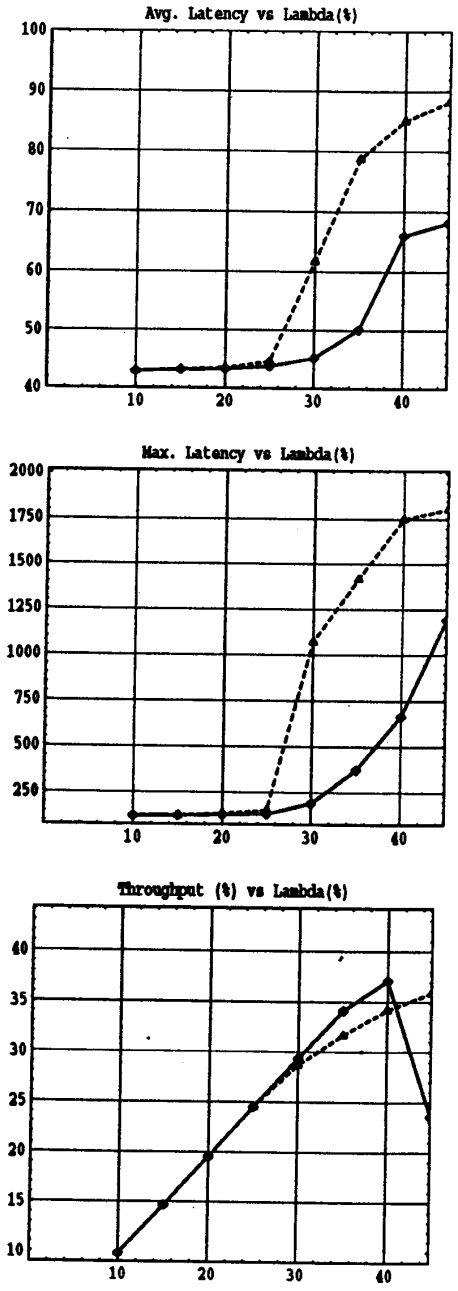


Fig. 12. Results for Transpose.

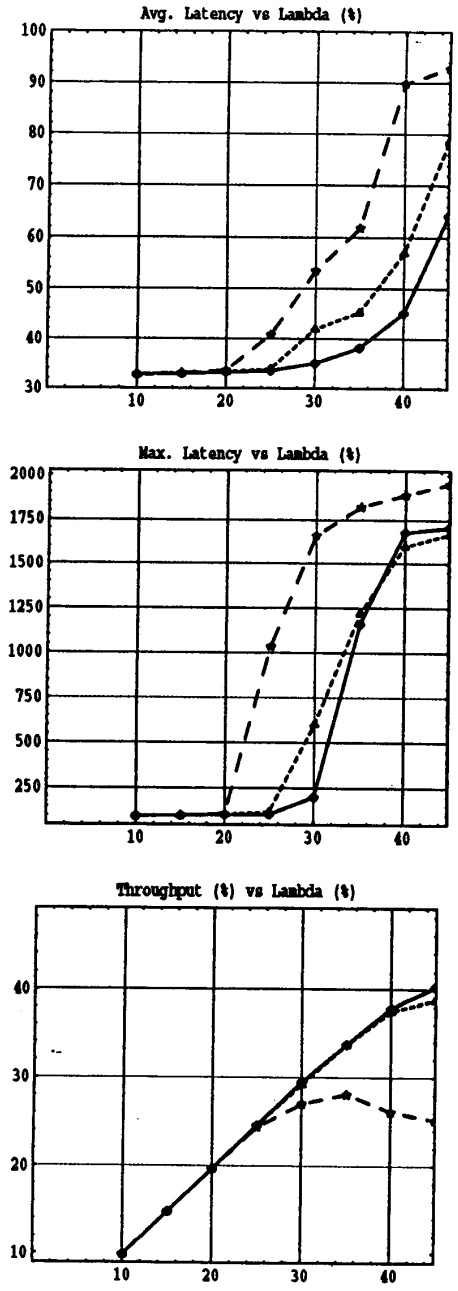


Fig. 13. Results for Bit-Reversal.

3) *Bit Reversal:* In this case, $c = 1/2$ again. As shown in Fig. 13, the maximum latency of the messages in *FULL* starts to grow quickly for λ higher than 30% of the maximum throughput, while the maximum latency of the messages in *ADAPT* and *OBLIVIOUS* starts to increase fast for λ higher than 25% and 20% of $\tau_{\max}$, respectively. Again, *FULL* suffers less than the other two algorithms from the applied loads and it degrades more gracefully.

As it is seen, *FULL* outperforms *OBLIVIOUS* under all traffic conditions and for all communication patterns.

VIII. Conclusion

In this paper, new algorithms for routing in hypercubes, two-dimensional meshes, and shuffle-exchanges are presented. These techniques have several appealing properties: they are deadlock and livelock free, fully-adaptive minimal (except for

the one for the shuffle-exchange), and a moderate amount of resources are necessary for their implementation.

The hypercube algorithm accomplishes fully-adaptive minimality while preserving deadlock and livelock freedom by using only two queues per routing node. This result improves on previous work in terms of the number of queues and the number of minimal paths available for routing. The mesh algorithm also provides the best known technique in terms of number of queues with the given properties of the routing algorithm. The algorithm for the shuffle-exchange is minimal in the sense that the number of exchange connections used for any given source-destination pair is the Hamming distance between these nodes. Furthermore, this technique is the first known in the literature to use a small (and independent of N) number of buffers and central queues in the design of the node.

An important tool on which the proposed methods are built is that of "hanging" an interconnection network. This idea has also been used in [21]–[24]. In this paper, it has been shown that this methodology is very useful for creating and visualizing routing functions. Furthermore, by hanging the underlying graphs, it is possible to generate queue utilization strategies that lead to dynamic deadlock-free conditions.

This paper showed the simulation of the performance of fully-adaptive minimal algorithms for deadlock- and livelock-free routing on hypercubes and two-dimensional meshes. Special attention was given to the mesh because of the potential practicality of this interconnection over the hypercube. Static and dynamic injection models have been tried. Several communication patterns have been used: random, complement, bit-reversal, transpose, and leveled permutations. These last four patterns are useful to test the ability of the routing algorithms to cope with the congestion arising in oblivious e-cube-type techniques. Furthermore, different loads have been applied to the routing techniques and critical network parameters such as average latency, maximum latency, and throughput have been measured.

The desirable properties that these routing techniques have in terms of deadlock-freedom and simplicity of the node are complemented with good routing performances, as shown by the simulations in Section VII. In particular, the fully-adaptive algorithms shown in this paper do not exhibit *any* congestion for *any* type of traffic, thus surpassing oblivious counterparts such as the e-cube or the $x - y$ algorithms.

Three algorithms called *FULL, ADAPT,* and *OBLIVIOUS* were tried on the two-dimensional 1K-node mesh. It has been shown that the routing algorithm *FULL* for the mesh has better throughput than *OBLIVIOUS* and *ADAPT*. These conclusions hold for the three types of communication patterns tried. An important conclusion from the results obtained for the fixed-permutation patterns is that network saturation occurs at substantially smaller values of applied load. For example, *FULL* accepts up to 30% of the maximum theoretical load for bit-reversal permutations while the same routing algorithm yields up to 75% of the maximum theoretical load for random routing. The simulations clearly show that for congesting communication patterns, as is the case for fixed-permutations, the *FULL* routing algorithm outperforms the others by sustaining a higher throughput.

In the hypercube, the results were also extremely satisfactory, for both static and dynamic injection. Random routing and three fixed-permutation patterns were tried. Oblivious routing techniques such as the e-cube algorithm are known to generate congestion when used on highly structured patterns such as routing-to-the-transpose. Therefore, only the *FULL* routing algorithm has been used in both static and dynamic injection models. No saturation took place in the hypercubes of up to 16K nodes for maximum loads, as a consequence of the rich connectivity of the network and the good performance of *FULL*.

Future work will be aimed at defining a much more detailed routing node model and flow control. Practical implementations of the principles shown in this paper are based on a switching mechanism that connects input to output buffers directly (i.e., not all switching is accomplished through the queues). With direct switching between input and output (both dynamic and static) buffers, only one queue per routing node is needed for increased throughput [39]. In addition, practical considerations call for a careful choice of the network topology. An attractive interconnection is the two-dimensional torus network. The torus offers an increased bandwidth at a relatively small expense measured in terms of the extra number of wires in the network when compared to the mesh. In [39], a fully-adaptive minimal deadlock- and livelock-free routing was devised for the 2-D torus. Excellent performance results were obtained with a similar approach to the one presented in this paper adapted to a more detailed node design for cut-through implementations [39].

APPENDIX

Theorem 2.1 [24]: The routing scheme proposed above over $\tilde{D}$, with the extended routing function $\tilde{\mathcal{R}}$, is deadlock free.

Proof: Let $FDQ(q)$ be the distance (in the underlying DAG D) from q to the farthest delivery queue reachable from q

$$\tilde{\mathcal{R}}(i_s, d_m) = \begin{cases} \{q_{A,s}\}, & \text{if } \exists j : s_j \neq m_j \text{ and } s_j = 0, \\ \{q_{B,s}\}, & \text{otherwise,} \end{cases}$$

$$\tilde{\mathcal{R}}(q_{A,n}, d_m) = \begin{cases} \{q_{A,\mathcal{E}^t(n)} : n_t \neq m_t\}, & \text{if } \exists j : n_j \neq m_j \text{ and } n_j = 0, \\ \{q_{B,n}\}, & \text{if } n \neq m \text{ and } \forall j : (n_j \neq m_j \Rightarrow n_j = 1), \\ \{d_m\}, & \text{if } n = m, \end{cases}$$

$$\tilde{\mathcal{R}}(q_{B,n}, d_m) = \begin{cases} \{q_{B,\mathcal{E}^t(n)} : n_t \neq m_t\}, & \text{if } n \neq m, \\ \{d_m\}, & \text{if } n = m. \end{cases}$$

through the underlying DAG D. It will be proved by induction on $FDQ(q)$ that no deadlock can arise.

- *Basis:* $FDQ(q) = 0$. Then, q must be a delivery queue. So, q is not deadlocked because it is not waiting for anything.
- *Inductive step:* Suppose now that $FDQ(q) = t$ and for all $t' < t$ the inductive hypothesis holds. Note that if $(q, q') \in A_s$, then $FDQ(q) > FDQ(q')$ because D is a DAG. So, the set $\{q' \in Q : (q, q') \in A_s\}$ (which is not empty, because of the fact that q is not a delivery queue) cannot have members in deadlock, because of the IH. So, as at least one successor of q in D belongs to $\tilde{\mathcal{R}}(q, d)$ (i.e., $\mathcal{R}(q, d) \neq \emptyset$), d being the destination of any message that can enter q, q will route any message in a finite amount of time. Thus, no deadlock can arise involving q.

The Hypercube Routing Function: In the following, $\mathcal{E}^i(k)$ is the number that has the same binary representation as k but for the ith digit, k_i. Formally, the routing function is as shown at the bottom of this page.

ACKNOWLEDGMENT

The authors wish to thank S. Konstantinidou for her many useful suggestions and comments. Also, they thank M. Lewis Fulgham for her suggestions and help with the software for the simulations.

REFERENCES

[1] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, pp. 547–553, May 1987.

[2] A. Ranade, S. Bhat, and S. Johnson, "The Fluent abstract machine," in *Fifth MIT Conference on Advanced Research in VLSI*, J. Allen and F. Leighton, Eds. Cambridge, MA: The MIT press, Mar. 1988, pp. 71–93.

[3] A. Ranade, "How to emulate shared memory," in *Foundations of Comput. Sci.*, 1985, pp. 185–194.

[4] E. Upfal, "An $O(\log N)$ deterministic packet routing scheme," presented at the *21st Annu. ACM-SIGACT Symp. Theory of Computing*, May 1989.

[5] T. Leighton and B. Maggs, "Expanders might be practical: Fast algorithms for routing around faults on multibutterflies," in *30th Annu. Symp. on Foundations of Comput. Sci.*, Oct. 1989, pp. 384–389.

[6] L. Valiant, "General purpose parallel architectures," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Amsterdam: North-Holland, 1988.

[7] S. Konstantinidou and L. Snyder, "The Chaos router: A practical application of randomization in network routing," in *2nd. Annu. ACM SPAA*, 1990, pp. 21–30.

[8] J. Ngai and C. Seitz, "A framework for adaptive routing," Tech. Rep. 5246:TR:87, Comput. Sci. Dep., Cal. Inst. of Technol., 1987.

[9] D. Hillis, *The Connection Machine*. Cambridge, MA: The MIT Press, 1985.

[10] F. Chong, E. Egozy, A. DeHon, and T. Knight, "Multipath fault tolerance in multistage interconnection network," Transit note #48, MIT, June 1991.

[11] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. Kung, *et al.*, "iWarp: An integrated solution to high-speed parallel computing," in *Proc. Supercomputing 88*, ACM, 1988.

[12] D. Lenoski, J. Landon, K. Gharachorloo, W. Weber, A. Goopta, and J. Hennessy, "Overview and status of the Stanford Dash multiprocessor," presented at the *Int. Symp. Shared Memory Multiprocessing*, Tokyo, Japan, Apr. 1991.

[13] C. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Trans. Comput.*, vol. C-32, pp. 1091–1098, Dec. 1983.

[14] S. Felperin, L. Gravano, G. Pifarré and J. Sanz, "Routing techniques for massively parallel communication," *Proc. IEEE* (special issue on massively parallel computers) vol. 79, Apr. 1991.

[15] M. Fulgham, R. Cypher and J. Sanz, "A comparison of SIMD hypercube routing strategies," RJ 7722 (71587), IBM Almaden Res. Ctr., 1990. (Also presented at the *Proc. ICPP '91, Int. Conf. Parallel Processing*, 1991.)

[16] L. Ni and P. McKinley, "A survey of routing techniques in wormhole networks," MSU-CPS-ACS-46, Dep. of Comput. Sci., Michigan State Univ., Oct. 1991.

[17] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A new computer communication switching technique," *Comput. Networks*, no. 3, pp. 267–286, 1979.

[18] N. Pippenger, "Parallel communication with limited buffers," in *Foundations of Computer Science*, pp. 127–136, 1984.

[19] T. Leighton, B. Maggs, and S. Rao, "Universal packet routing algorithms," 1988.

[20] D. Gelernter, "A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks," *IEEE Trans. Comput.*, vol. C-30, pp. 709–715, Oct. 1981.

[21] S. Konstantinidou, "Adaptive, minimal routing in hypercubes," in *6th. MIT Conf. Advanced Res. in VLSI*, 1990, pp. 139–153.

[22] Y. Birk, P. Gibbons, D. Soroker and J. Sanz, "A simple mechanism for efficient barrier synchronization in MIMD machines," RJ 7078 (67141) Comput. Sci., IBM Almaden Res. Ctr., Oct. 1989.

[23] K. Günther, "Prevention of deadlocks in packet-switched data transport system," *IEEE Trans. on Communications*, vol. com-29, April 1981.

[24] P. Merlin and P. Schweitzer, "Deadlock avoidance in store-and-forward networks. 1: Store-and-forward deadlock," *IEEE Trans. Commun.*, vol. 28, March 1980.

[25] W. Dally and H. Aoki, "Adaptive Routing using Virtual Channels," tech. rep., MIT, 1990.

[26] T. Leighton, "Average case analysis of greedy routing algorithms on arrays," in *SPAA*, 1990.

[27] P. Raghavan and E. Upfal, "A theory of wormhole routing in parallel computers," RJ:8512 (76733), IBM Res., Dec. 1991.

[28] G. Pifarré, L. Gravano, S. Felperin and J. Sanz, "Fully-Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks," in *Proc. the 3rd Annu. ACM Symp. Parallel Algorithms and Architectures*, 1991.

[29] R. Cypher and L. Gravano, "Adaptive deadlock-free packet routing in torus networks with minimal storage," RJ:8571 (77350), IBM Almaden Res. Ctr., Jan. 1992. (Also presented at the *ICPP'92*.)

[30] ——, "Requirements for deadlock-free, adaptive packet routing," tech. rep., IBM Almaden Res. Ctr., Feb. 1992. (Also presented at the *PODC'92*.)

[31] K. Bolding and L. Snyder, "Mesh and torus chaotic routing," UW CS91-04-04, Univ. of Washington, 1991. Also presented at the *MIT/Brown Advanced Res. in VLSI and Parallel Syst. Conf.*, Mar. 1992.

[32] W. Dally, "Virtual-Channel Flow Control," in the *17th Annu. Int. Symp. Comput. Architecture*, May 1990.

[33] D. Linder and J. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Trans. Comput.*, vol. 40, pp. 2–12, Jan. 1991.

[34] S. Felperin, L. Gravano, G. Pifarré, and J. Sanz, "Fully-adaptive routing: Packet switching performance and worm-hole algorithms," in *Supercomputing*, 1991.

[35] L. Gravano, G. Pifarré, S. Felperin, and J. Sanz, "Adaptive deadlock-free worm-hole routing with all minimal paths," TR: 91-21, IBM Argentina, CRAAG, Aug. 1991.

[36] P. Berman, L. Gravano, G. Pifarré, and J. Sanz, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks," tech. rep., IBM Almaden Res. Ctr., 1992. (Also presented in *ACM SPAA'92*, San Diego, CA.)

[37] B. Hajek and A. Greenberg, "Deflection routing in hypercube networks," *IEEE Trans. Commun.*, vol. 40, pp. 1070–1081, June 1992.

[38] B. Hajek, "Bounds on evacuation time for deflection routing," *Distributed Computing*, vol. 5, pp. 1–6, 1991.

[39] S. Felperin, H. Laffitte, G. Buranits, and J. Sanz, "Deadlock-free minimal packet routing in the torus network," TR: 91-22, IBM Argentina, CRAAG, 1991.

[40] G. Pifarré, L. Gravano, S. Felperin, and J. Sanz, "Fully-adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks," Tech. Rep., IBM Argentina, CRAAG, 1991.

[41] L. Valiant and G. Brebner, "Universal schemes for parallel communication," *ACM STOC*, pp. 263–277, 1981.

[42] L. Valiant, "Optimality of a two-phase strategy for routing in interconnection networks," Mar. 1982.

[43] E. Upfal, "Efficient schemes for parallel communication," *JACM*, vol. 31, pp. 507–517, July 1984.

[44] T. Leighton, D. Lisinski and B. Maggs, "Empirical evaluation of randomly-wired multistage networks," presented at the *ICCD'90*, 1990.

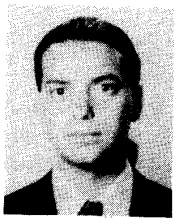
[45] S. Konstantinidou and E. Upfal, "Experimental comparison of multistage interconnection networks," RJ:8451 (76459), IBM Almaden Res. Ctr., Nov. 1991.
[46] R. Cypher and C. Plaxton, "Deterministic sorting in nearly logarithmic time on the hypercube and related computers," in *22nd. Annu. Symp. Theory of Computing*, 1990, pp. 193–203.
[47] D. Nassimi and S. Sahni, "Parallel permutation and sorting algorithms and a new generalized connection network," *JACM*, vol. 29, pp. 642–667, July 1982.
[48] J. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.*, vol. C-30, pp. 771–780, Oct. 1981.
[49] D. Dias and J. Jump, "Analysis and simulations of buffered delta networks," *IEEE Trans. Comput.*, vol. C-30, pp. 273–282, Apr. 1981.
[50] G. Pfister and V. Norton, "'Hot Spot' contention and combining in multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-34, pp. 943–948, Oct. 1985.
[51] J. Ngai and C. Seitz, "Adaptive routing in multicomputers," in *Opportunities and Constraints of Parallel Computing*, J. Sanz, Ed. New York: Springer Verlag, 1989.

Gustavo D. Pifarré was born in Buenos Aires, Argentina, in 1964. In 1985, he began studying computer science at the University of Buenos Aires (UBA), Argentina, and in 1988, he was admitted to the Escuela Superior Latino-Americana de Informatica (ESLAI) and graduated in July 1991. He is a teaching assistant and a Ph.D. student in computer science at UBA.

In 1990, he joined the Computer Research and Advanced Applications Group in IBM Argentina as a Researcher. He was twice a student visitor to the IBM Almaden Research Center, San Jose, CA. His current areas of interest include massively parallel routing and computers.

Luis Gravano was born in Buenos Aires, Argentina, in 1967. In 1986, he began studying computer science at the University of Buenos Aires (UBA), Argentina; in 1988, he was admitted to the Escuela Superior Latino-Americana de Informatica (ESLAI) and graduated in July 1991; and is now a Ph.D. student in the Computer Science Department at Stanford University, Stanford, CA.

In 1990–1992, he was with the Computer Research and Advanced Applications Group in IBM Argentina as a Researcher. He was a student visitor to the IBM Almaden Research Center, San Jose, CA on three occasions. His current areas of interest include databases and massively parallel routing.



Sergio A. Felperin was born in Buenos Aires, Argentina, in 1965. In 1984, he began studying computer science at the University of Buenos Aires (UBA), Argentina, and in 1988, he was admitted to the Escuela Superior Latino-Americana de Informatica (ESLAI) and graduated in July 1991.

In 1990, he joined the Computer Research and Advanced Applications Group in IBM Argentina as a Researcher. He was twice a student visitor to the IBM Almaden Research Center, San Jose, CA. He is also a teaching assistant in Computer Science at UBA. His current areas of interest include massively parallel routing, programming languages, and simulations.

Jorge L. C. Sanz (M'82–SM'86–F'91) was born in Buenos Aires, Argentina in 1955. He received the M.S. degree in computer science in 1977 and the M.S. degree in mathematics in 1978, and the Ph.D. in applied mathematics, working on complexity of algorithms, in 1981; all from the University of Buenos Aires.

He was with the University of Buenos Aires in the Department of Mathematics as an Instructor from 1978 to 1980 and conducted research as a scholar member of the National Council of Scientific and Technical Research of Argentina for four years. He was the recipient of many scholarships and a member of the Argentinian Institute of Mathematics. He was a visiting scientist at the Coordinated Science Laboratory, University of Illinois at Urbana-Cahmpaign, during 1981–1982. He was a visiting Assistant Professor at the Electrical Engineering Department and the Coordinated Science Laboratory in 1983. During that year, he was a summer visiting faculty member at the Computer Science Department in IBM Almaden Research Center, San Jose, CA, and from 1984 to 1993, he was a Research Staff Member. He conducts research work on industrial machine vision, parallel computing, and multidimensional signal processing. He was the technical manager of the machine vision group during 1985–86. He has also been an Adjunct Associate Professor with the University of California, Davis, where he conducted research as the Associate Director of the Computer Vision Research Laboratory until 1988. In 1994, he joined the Coordinated Science Laboratory and the Electrical and Computer Engineering Department at the University of Illinois at Urbana-Champaign as a Professor. His areas of broad professional interest are in computer science and applied mathematics. Specifically, he is interested in multidimensional signal and image processing, image analysis and machine vision, parallel processing, numerical analysis, computer architectures, and other subjects.

Dr. Sanz has served as a consultant of several companies in the US and is an active member of the ACM. In 1986, he received the IEEE Acoustic Speech and Signal Processing Society's Paper Award. He is a committee member of the Multidimensional Signal Processing Group of the IEEE Signal Processing Society. Dr. Sanz is an Associate Editor of the IEEE TRANSACTIONS ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING from August 1987 to August 1989. He was a Guest Editor of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 1988 special issues on Industrial Machine Vision and Computer Vision Technology. Also, he is Editor-in-Chief of *Machine Vision and Applications, and International Journal* (Springer-Verlag). He is the author of the book, *Radon and Projection Transform-Based Computer Vision* (Springer-Verlag, 1988) and *Massively Parallel Computing: Theory, Algorithms, Applications, and Technology* (Springer-Verlag, 1991). He is the Editor of the book, *Advances in Machine Vision* (Springer-Verlag). Dr. Sanz has been a chair and organizer of the 1988 IEEE Workshop on Machine Vision, held in Ann Arbor, MI. He has been the chair and organizer of the IBM Almaden-NSF Workshop on Opportunities and Constraints of Parallel Computing. He was the program committee chair of the VI IEEE Acoustic Speech and signal Processing Workshop on Multidimensional Signal Processing. He was the program committee chair of the Computer Architecture chapter at the 1990 International Conference on Pattern Recognition. He is the chairman of the Industrial Vision Chapter and a member of the Architecture Chapter of the International Association of Pattern Recognition.