Fully-Adaptive Routing: Packet Switching Performance and Wormhole Algorithms.

S.A. Felperin
CRAAG IBM Argentina
ESLAI
felperin@buevm2.vnet.ibm.com

L. Gravano
CRAAG IBM Argentina
ESLAI
gravano@buevm2.vnet.ibm.com

G.D. Pifarré
CRAAG IBM Argentina
ESLAI
pifarre@buevm2.vnet.ibm.com

J.L.C. Sanz
IBM Almaden Research Center
CRAAG IBM Argentina
sanz@ibm.com

Abstract

In this paper, a simulation study on the performance of some new algorithms for deadlock- and livelock-free adaptive routing is reported. Packet-switched fully-adaptive minimal routing on the mesh and the hypercube is explored for different injection models: static and dynamic. The algorithms simulated in this paper are the first known to be livelock- and deadlock-free fully-adaptive minimal that require a moderate amount of hardware. These techniques need only two central queues per routing node.

The performance of these schemes is measured for different traffic models: random, complement, transpose, bit-reversal and leveled permutations. Several machine sizes are tried and critical parameters indicating the performance of the routing algorithms are measured such as throughput, maximum and average latency, effective injection, and saturation point. In the case of the mesh network, the new method is compared to an oblivious scheme based on a similar routing node model. In the present version of this paper, simulation results are reported for hypercubes up to 16K nodes and for meshes of 1K nodes.

Finally, a fully-adaptive minimal worm-hole *routing algorithm for the torus network will be presented. This technique is deadlock- and livelock-free and requires only 8 virtual channels per physical bidirectional link for its implementation. Simulations are currently being performed to determine the practical performance of this routing method.*

1 Introduction and Definitions.

Deadlock freedom is one important property of routing algorithms. In order to avoid deadlock, many policies have been proposed [1, 2, 3, 4]. Generally, these policies are based on the idea of defining a partial order of the routing resources (either links or buffers) because deadlock can arise only if there exists cyclic waiting. This static restriction is too strong and can be weakened as shown in [2, 4].

Minimizing latency of routing schemes is an important objective in the design of interconnection networks. Routers that realize the minimal number of hops between the source and the destination of messages are appealing. Also, routing schemes that adapt the paths followed by messages according to local congestion encountered in the network are good candidates to reduce message latency and to provide higher throughputs. Livelock-freedom, deadlock-freedom, adaptivity, and minimality of routes have been realized in a single routing algorithm for hypercubes and meshes in [4]. This family of algorithms requires remarkably low hardware resources in the routing node.

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 [5, 6], [7], [8], [9]. Even when mathematical bounds can be estimated, the practical performance of routing algorithms is important. To this end, simulation results for static injection models have been reported by several authors [5] [10] [11].

On the other hand, deterministic bounds have been proved for sorting-based algorithms [12] and for adaptive algorithms based on multibutterfly construction [13]. In 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 [14] [15] [16] [17]. 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.

This work presents a number of simulation results on the routing techniques for the mesh and the hypercube presented in [4]. The simulation work is aimed at showing the practicality of the new routing ideas for different communication patterns and models of injection. Some comparisons are made with other routing algorithms [18].

In same cases, messages are too big to be stored completely within a one routing node. In this situation different routing algorithms and node models are necessary. One approach is known as *wormhole* routing [3].

In this paper, a fully-adaptive minimal *worm-hole* routing algorithm for the torus network is presented. This technique is deadlock- and livelock-free and requires a very moderate amount of resources in the routing nodes [19]. The algorithm for the torus is the first known to be fully-adaptive minimal involving only 8 virtual channels per physical link.

The organization of this paper is as follows. In Section 1.1 the packet routing algorithms are presented, followed by the node in Section 1.2. Section 2 presents the assumptions made for the simulations, node activity, injection policy and communication patterns. Sections 3 and 4 present the results of the simulations for the hypercube and the mesh, respectively. Section 5 will present the wormhole algorithm for the torus network. Section 6 will present some conclusions.

1.1 The routing model.

In packet routing, the critical resources are the queues used to store the messages during their way towards their destinations. 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.

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 routing function will be expressed in terms of the queues of each node. To do so, each queue will have an identifier that distinguishes it from the rest of the queues. The set of delivery queues of all the network will be referred to as $DelivQ$. Notice that each delivery queue identifies a unique node of the network. The set of injection queues will be referred to as $InjectQ$. Each message has a destination associated with it, given by the function $Dest$: Messages $\rightarrow$ $DelivQ$.

A total routing function $\mathcal{R}$: $Queues \times DelivQ \rightarrow \mathcal{P}(Queues)$ 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 ($q_i, q_j \in Q$) iff there exist an injection queue s and a delivery queue d such that $\mathcal{R}$ builds a route from s to q 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 [3] 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. Let $d^-_{(Q,A_s)}(q) \triangleq \{q' \in Q : (q,q') \in A_s\}$ be the set of *direct successors* of q. Whenever there is no ambiguity, the subscripts will be dropped.

Every non-delivery 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. $Level(q)$ 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 [2].

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. Now, let $\tilde{D} = (Q, A_s \cup A_d)$ 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 link set* and A_d will be called the *dynamic link 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. If $q' \in \tilde{\mathcal{R}}(q,d)$, then $(q,q') \in A_s \cup A_d$.
2. $\mathcal{R}(q,d) \subseteq \tilde{\mathcal{R}}(q,d)$.
3. 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 link, it will still have the possibility of taking a static link as a next step towards its destination. Therefore, at any moment, every message has a static-link 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.

In [4] the following routing idea has been proposed for the hypercube. Level each node with its hamming weight and divide the routing of each packet into two phases. In its first phase, a message can be moved from node s to a neighboring node d on a *minimal path* towards its destination either if the level of d is greater than the level of s (this is a *static* move) or if the level of d is smaller than the level of s and the message has *at least one more increasing-level step* to

Figure 1: The QDG for the 3-hypercube.

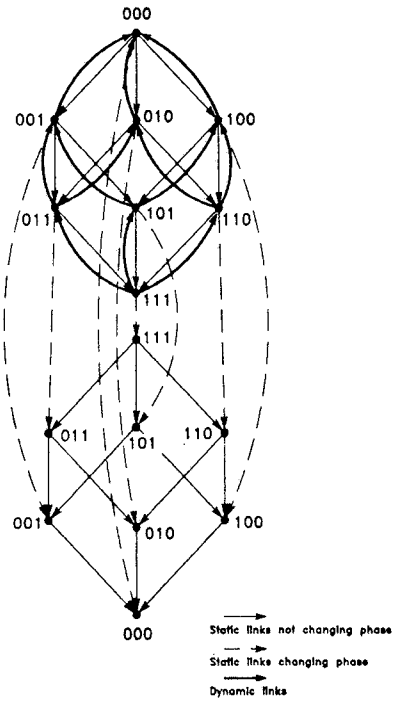
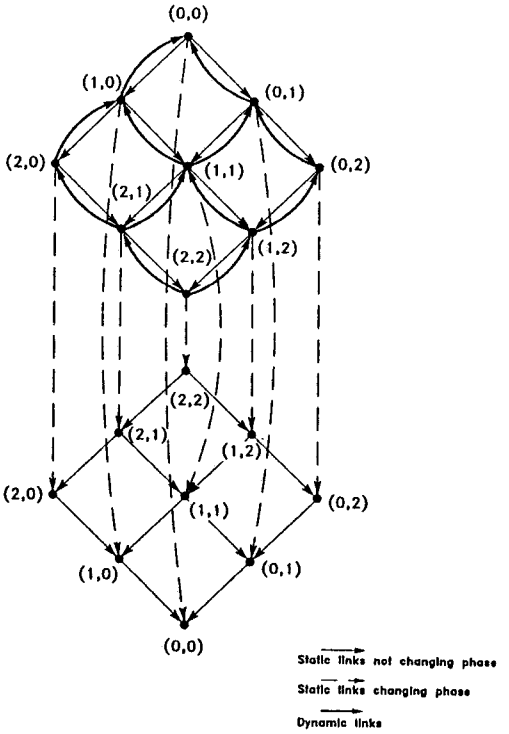


Figure 2: The QDG for the mesh



make from d (this is a *dynamic* move). Once a message has no more increasing-level steps to make, it begins its second phase, in which all moves are decreasing-level (the moves of the second phase are all static).

A similar routing algorithm can be developed for the mesh if node (x, y) is leveled with $x + y$ instead of the hamming weight. Figures 1 and 2 show the QDGs for the hypercube and the mesh, respectively.

It should be noted that these routings are fully-adaptive, minimal, and deadlock-free. The deadlock-freedom can be proved using the fact that the static moves form a DAG and the dynamic ones satisfy the restrictions described above. See [4] for details.

Throughout this paper, the routing just described will be called *FULL*. The routing that results from removing all the dynamic moves from *FULL* will be called *ADAPT*. This routing is still adaptive and minimal but not fully-adaptive as some possible minimal paths are excluded. The routing that results from choosing one successor from the DAG in a deterministic way will be called *OBLIVIOUS*. This last routing has no adaptivity at all.

1.2 The model for the routing node.

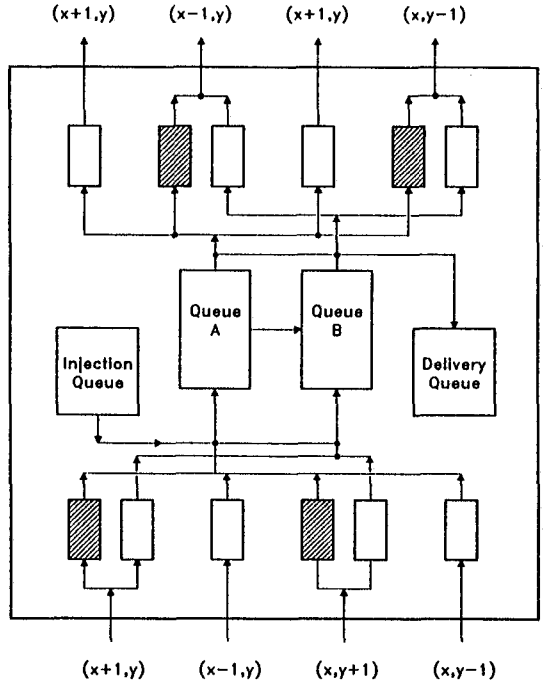
Functional descriptions of the routing nodes needed to implement the algorithms in the mesh and the hypercube were presented in [4]. This model, based on the one presented in [20] for the hypercube, will be described here briefly.

As described in Section 1.1, a message can move from a queue to another queue following two types of transitions, either through dynamic links or through static links. There exists a dynamic or static link between a pair of queues only if these queues are at distance at most one in the physical network, i.e. if either the queues are in the same node or at adjacent nodes (nodes connected by a physical link in the network).

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, modelling 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 its 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. The idea is that the queue size will be independent of the network size.

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 links and those with static links. Consider link j, incident to nodes n and n'. If traffic corresponding to dynamic links 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, when a message wants 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 and it will arrive at the input buffer in node n that is

Figure 3: The node for the Mesh.



associated with both dynamic transitions and link j. If traffic corresponding to static links 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'.

This node model is illustrated for the mesh in Figure 3. The cross-hatched boxes correspond to buffers associated with dynamic links. A similar node is obtained for the hypercube (see [4]).

Possible hardware implementations require only one central queue and switching between input and output buffers.

2 The simulations.

2.1 Network activity.

This section describes the activity of the network in order to run the algorithms described in Sections 1.1 and 1.2. As latency will be measured in *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, each node is able to scan all its input buffers and its injection buffer and move the messages to the corresponding queue, provided there is room for them. 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 room in a central queue for its message in the previous cycle. Once this step has been done, the node is allowed to inject a message in its injection buffer, provided it is empty and the node has a message to inject. Next, the queues send their messages to a suitable output buffer. The queues are scanned in FIFO order, and all the messages that find their corresponding output buffer empty are allowed to move there. 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[1] 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.

2.2 Injection model.

There are two kind of 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 begins when each node injects its first message, and ends when the last message arrives at its destination. The interesting parameters of this kind of injection are L_{max}, the maximum latency, and L_{avg}, the average latency.

In addition, in the second model it is worthwhile to measure I_r, the effective injection rate: the number of times a node succeeded in injecting relative to the number of trials, and the true injection $I_t = I_r\lambda$, where λ is the average number of packets that a node tries to inject per cycle. Note that I_t is related to the network throughput. If the throughput S is defined as the average number of messages that are injected in a cycle, then $S = NI_t$. So, I_t is the throughput normalized by the network size.

The dynamic injection model presents a number of phenomena that do not show up in the static case. The most obvious one is that dynamic injection leads to infinite processes. So, in order to observe the interesting phenomena the process has to be truncated at some point. Another problem is that if λ is too large, the system can be *saturated*, i.e. L_{max} grows without bound. So, the maximum λ that maintains the system unsaturated is a key parameter to measure.

Some simple bounds are known for λ. If N is the number of nodes of the network, B the bisection of the network [9] [21] and c the proportion of messages that cross the bisection, then

$$\frac{N\lambda c}{2} < B \qquad (1)$$

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

This formula means that, in every cycle, the average 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

$$\lambda < \frac{2B}{Nc} = \lambda_{max} \quad (2)$$

2.3 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 messages with destination $\sigma(p)$. 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, the transpose means that 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 $n \times n$-mesh, this means that node (x, y) will send messages to node $(n-1-x, n-1-y)$[2]. 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 Section 1.1, each node has a level associated. 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.

3 The simulations for the mesh.

In this section, the simulation results for the mesh will be presented. In this topology, the algorithms *FULL*, *ADAPT* and *OBLIVIOUS* behave in the same way for static injection. Therefore, only the results for dynamic injection will be considered here. Recalling Equation 2, it should be noted that in an $n \times n$-mesh, $B = n, N = n^2$, so $\lambda_{max} = 2/(nc)$. This means that λ decreases very rapidly with n. The simulations focused on only one network size, i.e. 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. Note that this restriction is not related to the deadlock-freedom property of the algorithm because this property is proved by using only the fact that a message has always a static path to its destination. Thus, the restriction on the use of dynamic transitions can only affect the performance of the *FULL* algorithm.

Readers should note that after the network saturates, the values in the plots shown below have no meaning at all.

For all the communication patterns simulated, there are plots for L_{avg}, L_{max}, I_r (%), and I_t as a function of λ/λ_{max}, i.e. the proportion of the maximum theoretical injection allowed in each case. Simulations were performed for injection rates going from 10% of λ_{max} to its 80%, in intervals of 5%.

In the following plots, the whole 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.

Due to space constraints, only the plots for Random Routing and Transpose will be shown here. In [18] the complete plots are presented.

The results are the following:

1. Random Routing. In this case, $c = 1/2$, so $\lambda_{max} = 4/n = 0.125$. Figure 4 shows that the algorithm *FULL* was able to accept injection up to the 75% of λ_{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. Figure 4 also shows that for *FULL* the maximum I_t (and then the maximum throughput) is reached when $\lambda = 75\%\lambda_{max}$, $I_t = 3/n = 0.09375$.
2. Complement. Here, $c = 1$, so $\lambda_{max} = 2/n = 0.0625$. The *FULL* algorithm accepted injection up to 60% of λ_{max} before saturation, *ADAPT* accepted up to 50% and *OBLIVIOUS* accepted 45%. For *FULL*, I_t can grow up to $1.2/n = 0.0375$ [18].

   The most interesting difference between Random Routing and Complement (this difference will be preserved in the following two patterns) is that

[2] It is supposed that the nodes are numbered from $(0, 0)$ to $(n-1, n-1)$.

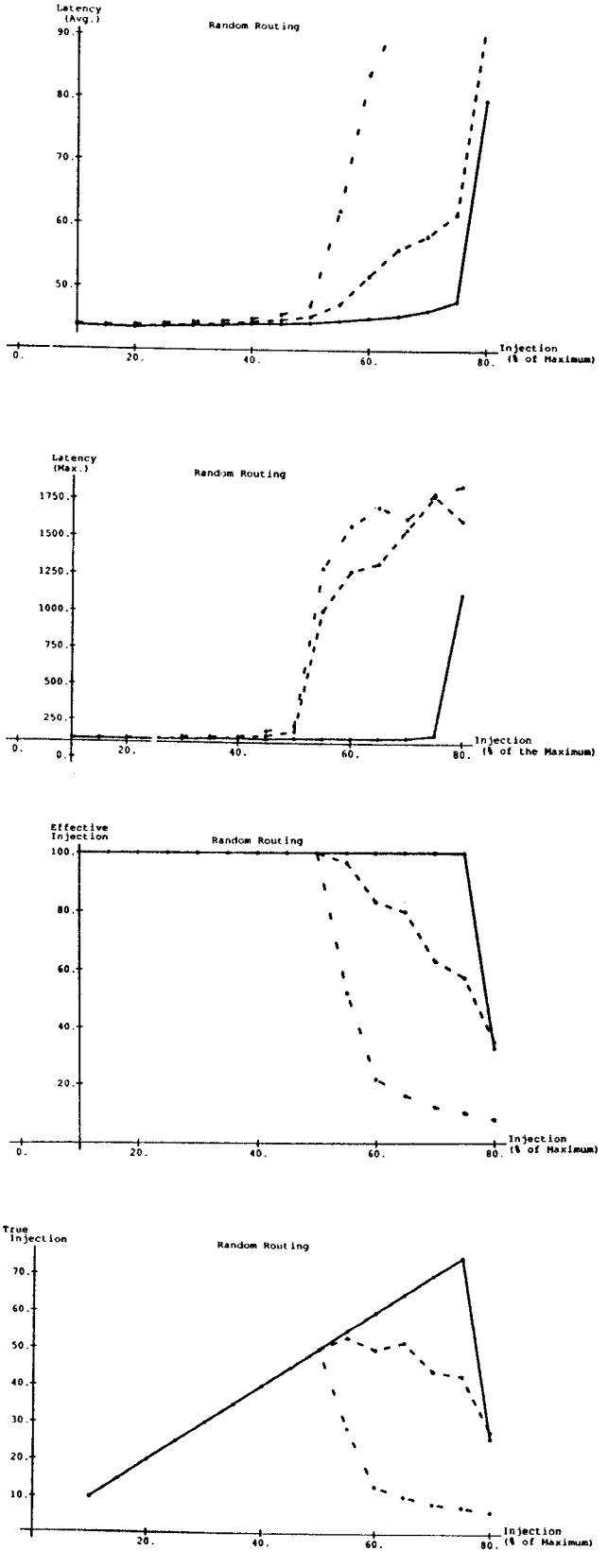


Figure 4: Results for Random Routing.

Table 1: 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

Complement saturates before Random Routing. This is based on the fact that the structured patterns are more congesting that the random ones.

3. Transpose. Here, $c = 1/2$ again, and λ_{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*.

   Figure 5 shows that the *FULL* algorithm stands injection up to 35% of the maximum, while *OBLIVIOUS* (and *ADAPT*) up to 25%. Then, as shown in Figure 5 for *FULL*, I_t can be up to $1.4/n = 0.04375$.

4. Bit Reversal. In this case, $c = 1/2$ again. *FULL* accepted injections up to 30% of λ_{max}, *ADAPT* accepted up to the 25% and *OBLIVIOUS* does so for up to 20%. For *FULL*, I_t was up to $1.2/n = 0.0375$ [18].

4 The simulations for the hypercube.

In this Section, simulations results of the routing algorithm proposed will be shown for the hypercube. Recalling Equation 2, it should be noted that for the n-dimensional hypercube, $B = 2^n/2 = N/2$, so $\lambda_{max} = 1/c$. This means that λ does not depend on the size of the network. Note that since c is a proportionality factor, $c \leq 1$, and then $\lambda \geq 1$. 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 1 to 4 show the results of static injection and Tables 5 to 8, the results for dynamic injection. It should be noted that no saturation arises in the hypercube, but I_r (%) degrades. This is due to the fixed size of the queues.

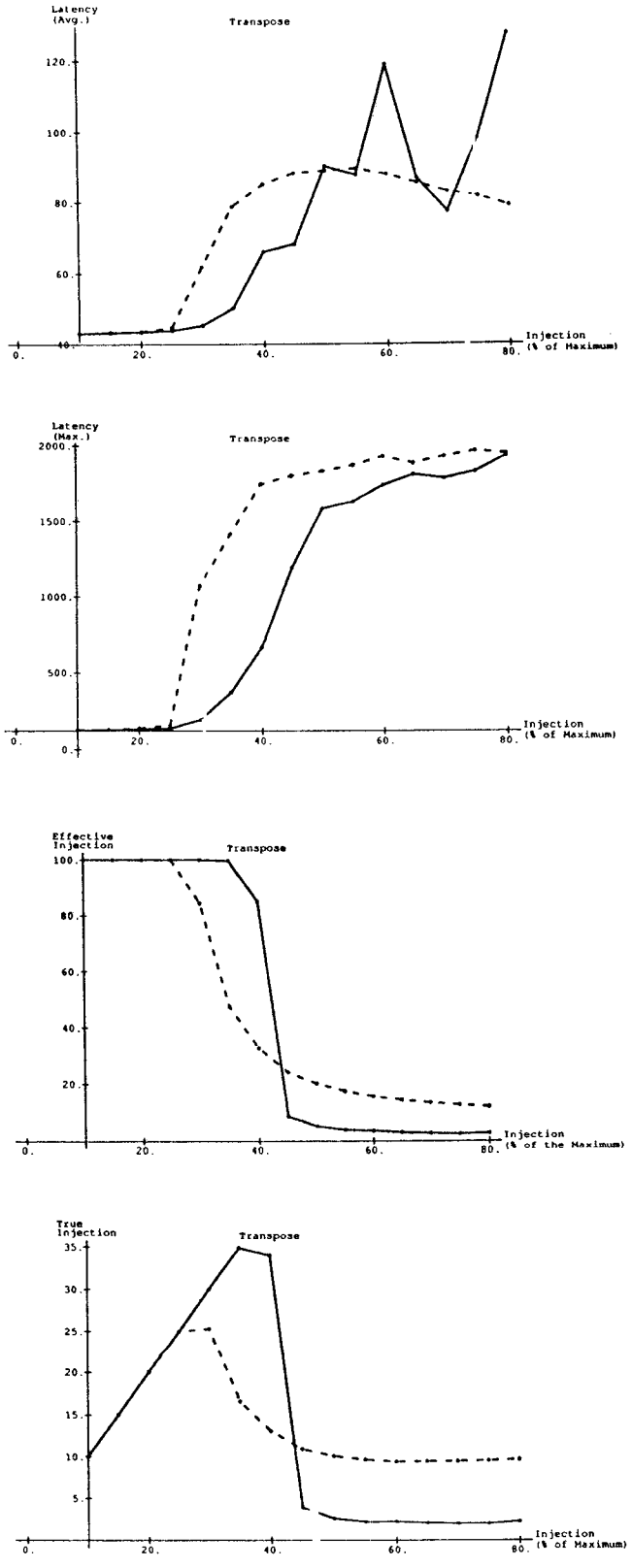


Figure 5: Results for Transpose.

Table 2: 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 3: 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 4: 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

Table 5: Random Routing, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
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 6: Complement, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
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 7: Transpose, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
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 8: Leveled Permutation, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
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

5 Fully-adaptive minimal worm-hole routing in Tori.

In this Section, a fully-adaptive, minimal, deadlock-free worm-hole routing algorithm for the torus network will be described. This algorithm requires only a moderate amount of resources for its implementation. 8 virtual channels (see [3]) per bidirectional physical link are required, while allowing all the minimal paths between any pair of nodes s, d to be of potential use by those messages moving from s to d. In [22], a fully-adaptive, minimal, deadlock-free worm-hole routing algorithm for the torus network has been presented, but it requires that some links have 12 virtual channels associated with them. The routing algorithm that will be presented in this Section uses fewer virtual channels than the one in [22].

Each of the two dimensions of the network, namely X and Y, has two possible *orientations*: X^+ or X^-, and Y^+ or Y^-, respectively. Node (x, y) is connected to node $(x+1 \bmod k, y)$ following orientation X^+, and to node $(x-1 \bmod k, y)$ following orientation X^-. The connections following orientations Y^+ and Y^- are defined analogously.

If for a given node (x, y) dimension Y is fixed to y, then a length- k cycle is defined by moving along the X dimension. Cycles along dimension Y are defined in an analogous manner. Given a message with source node (x, y) and destination node (x', y'), a minimal path from (x, y) to (x', y') is built in such a way that the message will have to travel through at most $\lfloor k/2 \rfloor$ links along dimension X and through at most $\lfloor k/2 \rfloor$ along dimension Y. To find such minimal paths, the correct orientation along each dimension should be chosen. This choice is independent for each dimension. In this paper, only the case k odd will be considered. The case k even presents certain subtleties that will not be dealt with here for the sake of clarity[3]. Consider dimension X and the set of cycles associated with this dimension. Moving along dimension Y does not change the relative position within the different cycles associated with dimension X that will be visited. Similar considerations can be made for dimension Y. Therefore, the set of minimal paths between any pair of nodes is determined by a correct choice of the direction in which to change each of the dimensions.

Consequently, when a message is injected in the network, it will be classified into one out of 4 classes according to the orientation in which each dimension should be corrected in order to follow a minimal path towards the destination. These 4 classes will be referred to as X^+Y^+, X^-Y^+, X^+Y^-, and X^-Y^-[4]. Given a source node, a destination node, and an orientation for each dimension, a "submesh" is determined. All minimal paths between a source and a destination

[3] If k is even, and a message m is $k/2$ steps away from its destination along dimension X, for example, then either orientation along the X dimension can be taken in order to follow minimal paths towards m's destination.

[4] Again, only the case of k-tori with k odd will be considered here. If k is even, some messages may belong to more than one class, depending on the final destination of the messages.

node are included in the submesh determined by the source, the destination and the orientation of each dimension chosen as explained above.

Now, the set of virtual channels associated with each of the 4 classes into which messages are classified, viz X^+Y^+, X^-Y^+, X^+Y^-, X^-Y^-, will be presented. Each of these classes will have a particular set of virtual channels assigned. There will be 2 virtual channels per bidirectional physical link for each of the classes. The definition of each of these 4 sets of virtual channels is almost identical. So, only the case X^+Y^+will be described here.

A message will belong to class X^+Y^+if the minimal paths from its source towards its destination are built by choosing X^+ and Y^+ as the orientation for correcting dimensions X and Y, respectively. Only this kind of messages will be routed through the virtual channels associated with class X^+Y^+.

Each physical link will have two virtual channels associated in the set of virtual channels corresponding to the class X^+Y^+. These two virtual channels will both use the underlying bidirectional physical link in the same direction. Every virtual channel is identified by a 3-uple: the first component of the 3-uple is either a 0 or a 1 [5]. The second component is the dimension associated with the underlying link: a 0 indicates the X dimension, and a 1, the Y dimension. Finally, the last component is the node in which a message arrives by taking this virtual channel. For example, there are two virtual channels from node $(3,2)$ to node $(4,2)$: $c_{0,0,(4,2)}$ and $c_{1,0,(4,2)}$.

Next, the routing function for routing on the k-torus used for those messages belonging to class X^+Y^+will be described. The routing function will be defined in terms of the virtual channels just mentioned. Let m be a message with source node (x,y) and destination node (x',y'). A distinction has to be made between those messages that will need to use the wrap-around links and those that will not. If m does not have to go through any wrap-around, then m will visit only channels with prefix 1 until it arrives at its target node. At each step, m will move along any of the dimensions that need correction. If $x > x'$ then m will use a wrap-around corresponding to dimension X and orientation X^+. If $y > y'$ then m will use a wrap-around corresponding to dimension Y and orientation Y^+. Again, the use of wrap-arounds or not along dimension X (resp. Y) depends exclusively on the values of x and x' (resp. y and y'). If either $x > x'$ or $y > y'$, then m will start moving through virtual channels with prefix 0 again, moving along any of the dimensions that need to be corrected, until it has to go through a wrap-around. It is important to notice, as will be emphasized below, that those virtual channels $c_{0,d,(z,w)}$ with $d = 0$ or $d = 1$, and $0 \leq z, w \leq \lfloor k/2 \rfloor$ are not used during this first phase. This is so because if $x > x'$ and $x < \lfloor k/2 \rfloor$ then the path from x to x' following X^+ has length greater than $\lfloor k/2 \rfloor$ and so, the X^- orientation would have been chosen to correct dimension X. An identical argument follows for dimension Y.

After using a wrap-around, m will start moving through virtual channels with prefix 1. m will go on moving through channels with prefix 1 either until it arrives at its destination node, or until it needs to use another wrap-around, corresponding to the dimension that has not gone through a wrap-around yet. At this moment, m will start visiting channels with prefix 0 again until it arrives at its target node. It is important to note that the channels with prefix 0 that m will use in its last phase towards its destination are exactly those channels that have not been used yet, as pointed out just above.

The routing algorithm that has just been described allows each message to choose adaptively *any* one among *all* the minimal paths from its source node to its destination node. Furthermore, the routing algorithm is deadlock-free.

It is supposed that every node (x,y) has an *injection channel* $c_{(x,y)}$ into which node (x,y) injects its messages.

Theorem 5.1 *The routing algorithm presented in Section 5 is correct and deadlock-free.*

Proof: The proof can be found in [19].

6 Conclusions.

This paper showed the performance of fully-adaptive minimal algorithms for deadlock- and livelock-free routing on hypercubes and 2-dimensional meshes. Static and dynamic injection models have been tried. Several communication patterns were used: random, complement, bit-reversal, transpose, and leveled permutations. These last four patterns are useful to test the ability of the routing to cope with potential congestion arising in practical routing. Furthermore, different loads were applied to the routers and critical network parameters such as average latency, maximum latency, effective injection rate, true injection, and throughput were measured.

The desirable properties that these routers have in terms of deadlock-freedom and simple architecture of the node are complemented with good routing performances, as shown by the simulations in this paper.

Three algorithms called *FULL*, *ADAPT*, and *OBLIVIOUS* were tried on the 2-dimensional mesh for up to 1K nodes. These algorithms exhibit different amounts of adaptivity to congestion. It has been shown the router *FULL* for the mesh has a better saturation point and throughput than *OBLIVIOUS* and *ADAPT* while similar latency is achieved by the three methods. These conclusions hold for the four types of communication patterns tried. These simulations provide some evidence on the better performance of the proposed adaptive algorithm for the machine size 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 router yields up to the 75%

[5] This first component will often be referred to as the *prefix* of the virtual channel in what follows.

of the maximum theoretical load for random routing. The simulations clearly show that in cases where more congestion is present in the network (as in the case of fixed-permutations) the *FULL* router outperforms the others by allowing a higher throughput.

In the cube, the results were extremely satisfactory, for both static and dynamic injection. Random routing and three fixed-permutation patterns were tried. Oblivious routers may generate congestion when used on highly structured patterns such as routing-to-the-transpose. Therefore, only the *FULL* router was used in both static and dynamic injection models. No saturation happened in the hypercubes of up to 16K nodes for $\lambda = 1$, as a consequence of the rich connectivity of the network and the good performance of *FULL*.

The worm-hole routing algorithm for the torus presented in Section 5 is the first known to be fully-adaptive minimal deadlock- and livelock-free involving only 8 virtual channels per physical link. Simulations are being performed to determine the practical performance of this routing method.

Acknowledgements.

The authors wish to thank to Smaragda Konstantinidou, for her many suggestions and comments. Also, to Melanie Lewis Fulgham for her suggestions and help with the software for the simulations. This work was developed in the IBM Almaden Research Center.

References

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

[2] P. Merlin and P. Schweitzer, "Deadlock avoidance in store-and-forward networks. 1: Store-and-forward deadlock," *IEEE Transactions on Communications*, vol. 28, March 1980.

[3] W. Dally and C. Seitz, "Deadlock-free routing in multiprocessor interconnection network," 5206:TR:86, Computer Science Department, California Institute of Technology, 1986.

[4] G. Pifarré, L. Gravano, S. Felperin, and J. Sanz, "Fully-Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks," in *3rd. Annual ACM SPAA*, 1991.

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

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

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

[8] A. Ranade, "How to emulate shared memory," in *Foundations of Computer Science*, pp. 185 – 194, 1985.

[9] T. Leighton, "Average Case Analysis of Greedy Routing Algorithms on Arrays," in *SPAA*, 1990.

[10] M. Fulgham, R. Cypher, and J. Sanz, "A comparison of SIMD hypercube routing strategies," RJ 7722 (71587), IBM Almaden Research Center, 1990.

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

[12] R. Cypher and C. Plaxton, "Deterministic sorting in nearly logarithmic time on the hypercube and related computers," in *22nd. Annual Symposium on Theory of Computing*, pp. 193–203, 1990.

[13] E. Upfal, "An $\mathbf{O}(\log N)$ deterministic packet routing scheme," in *21st Annual ACM-SIGACT Symposium on Theory of Computing*, May 1989.

[14] J. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, vol. c-30, pp. 771–780, October 1981.

[15] C. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Transactions on Computers*, vol. c-32, pp. 1091–1098, December 1983.

[16] D. Dias and J. Jump, "Analisys and simulations of buffered delta networks," *IEEE Transactions on Computers*, vol. c-30, pp. 273–282, April 1981.

[17] G. Pfister and V. Norton, "'hot spot' contention and combining in multistage interconnection networks," *IEEE Transactions on Computers*, vol. c-34, pp. 943–948, October 1985.

[18] S. Felperin and J. Sanz, "Simulation Results on the Performance of Fully-Adaptive Minimal Deadlock-Free Routing for Meshes and Hypercubes," TR:91-06, IBM Argentina, CRAAG, March 1991.

[19] L. Gravano, G. Pifarré, and J. Sanz, "Adaptive Worm-hole ' ıting in Tori and Hypercubes," TR:91-1 .BM Argentina - CRAAG, March 1991.

[20] S. Konstantinidou, "Adaptive, minimal routing in hypercube," in *6th. MIT Conference on Advanced Research in VLSI*, pp. 139–153, 1990.

[21] J. Ngai and C. Seitz, "Adaptive routing in multicomputers," in *Opportunities and constraints of parallel computing* (J. Sanz, ed.), Springer Verlag, New York, 1989.

[22] D. Linder and J. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, vol. 40, pp. 2–12, January 1991.