The Chaos Router: A Practical Application of Randomization in Network Routing

S. Konstantinidou and L. Snyder*
Dept. of Computer Science and Engineering
University of Washington

Abstract

We present the chaos router, an asynchronous adaptive router, which under certain circumstances can send messages farther from their destinations. The chaos router greatly simplifies the routing logic by removing the livelock protection of previous schemes. Through an effective use of randomness, whose sources include that due to the adaptively processed load, the natural timing differences of selftimed circuitry and explicitly injected randomization, the chaos router avoids long message routes with high probability. In this paper the router is described, it is argued that the chaos router is deadlock free and probabilistically livelock and starvation free, and simulation results are presented showing that the chaos router performs well.

Introduction

The problem of message routing in multicomputers, where processors communicate and synchronize by exchanging messages, has long been studied. In this model of computation routing is assumed to be *local*, where routing nodes have no knowledge of the traffic load in other nodes, and *continuous*, where nodes inject messages independently of each other and without knowledge of the state of the network as a whole.

State-of-the-art routers for MIMD machines such as the iPSC, NCUBE and Ametek implement *oblivious* routing [1], where a message's path is completely determined by the (source-address, destination-address) pair. Oblivious routers require only relatively simple logic in order to route messages and to guarantee the three essential properties of every router *i.e.* freedom from deadlock, livelock and starvation. As a result, oblivious routers can be very fast under light to moderate random traffic [2, 3]. However, they have an $\Omega(\sqrt{N}/d)$ worst case routing delay in any N node, degree d network [4] and they are fault intolerant.

Among proposed alternatives are randomized routers, of which Valiant and Brebner's [5] is an instance. It requires that messages be first routed from their source to a randomly selected intermediate node and from there to their destination. This router has been analyzed and shown to have $O(logN)$ expected time. The practical problem with this router is that it doubles the length of the average message path. Though negligible in theory, this factor of two is unacceptable in practice. Moreover, the router penalizes "average random traffic" to improve the worst case traffic, which is presumably rarer.

Adaptive routers, another alternative to obliviousness, select message paths based on the local load characteristics. Thus, adaptive routers can diffuse local congestion by exploiting alternative paths to a destination; they are also more *fault tolerant*, a consideration that becomes increasingly important as systems get larger. *Minimal* adaptive routers, where messages are always routed closer to their destination, have been proposed [6]. However, our simulations show that *nonminimal* adaptive routers, where messages are permitted to be *derouted* or *misrouted*, *i.e.* sent further from their destination in the presence of congestion, are better at handling non-uniform traffic. Examples of

*This work was supported in part by the Defense Advanced Projects Agency under Contract N00014-88-K-0453, in part by the Office of Naval Research under Contract N00014-89-J-1368 and in part by an IBM Graduate Fellowship.

adaptive, nonminimal routers include Hillis' Connection Machine router [7], the Ngai and Seitz [2] router and the Shuffle-Exchange router suggested in [8]. In the class of adaptive, nonminimal routers one can also include Gelernter's router [9], where nodes can occasionally *refuse* messages and Grunwald's [10] backtracking circuit switched router.

Though potentially fast and robust, nonminimal routers are subject to *livelock*, a serious problem wherein messages fail to be delivered because they are repeatedly derouted. The "standard" * solution to livelock is to timestamp every message; when multiple messages require the same channel, the oldest is selected; eventually any message "ages" enough to be delivered. The problems with this *prioritized* solution to livelock and its variants are (1) the timestamp, which must be sufficiently large not to overflow, adds bits to the message, and (2) most critically *the process of selecting the oldest among messages competing for a channel complicates the routing decision.* It adds gate delays, slows the router and would seem to prejudice the chances that adaptive routers could replace oblivious routers.

The *chaos router*, proposed here, is an asynchronous adaptive, nonminimal router that *eliminates* the need for livelock protection. The intuition is that the router exploits randomness to keep the network "chaotic," thereby causing the repetitious routing patterns, typical of livelocked messages, to decay.

The principal new ideas in the chaos router are as follows. First, the chaos router is fully asynchronous; not only do nodes operate independently of each other but even channels operate independently of each other and the router's internal logic. This asynchrony makes all of the routers effectively independent of one another and it should be contrasted with synchronous routers like the CM [7] and with other asynchronous routers like the Ngai and Seitz [2] router where a variety of dependences exist within the router and between routers.

Second, the circuitry of the routers is selftimed, *i.e.* not clocked. Thus the routers will run at different rates because of differences in the tasks being performed as well as variations in the electronics. Though selftiming has been used before, the importance here is that because of the asynchrony just mentioned these timing differences propagate and lead to differences in when and how the routers perform various activities. These timing differences are in addition to the behavioral differences engendered by the dynamic characteristics of the load.

Third, the chaos router explicitly randomizes its message selection during derouting. This means that unlike other nonminimal routers the chaos router guarantees a nonzero probability that a message can avoid derouting. We should note that the concept of randomly selecting which message to deroute was also suggested in [8, 11, 12] although in an entirely different context. In particular, the context was message switching on a synchronous shuffle-exchange network and its variants (multiple-layer, multiple-stage, buffered). In all these cases, when necessary, one of the two conflicting messages at the inputs of a switch was randomly selected for derouting. Livelock issues were not considered.

The principal effect of this chaos, besides assuring network unpredictability and removing the overhead of livelock protection, is to make the router probabilistically livelock free, *i.e.* the probability of a message remaining in the network longer than t seconds goes to zero as t increases. In practice, as the simulations show, probabilistic livelock freedom can be considered to be operationally equivalent to absolute livelock freedom.

In this paper we describe the chaos router. We abstract chaotic routing and argue that chaos routers are deadlock free and probabilistically livelock and starvation free. Finally, we present simulations illustrating the performance of a 256 node chaotic network. The chaotic network always runs faster than the prioritized network and exhibits substantially less derouting.

An asynchronous, randomized, adaptive router

In this section, we will present our routing algorithm. For the presentation we assume a binary hypercube topology, although the same routing concepts can be used with any k-ary d-cube.

We assume that our router implements an asynchronous computational model. Each node operates at its own rate. It communicates with its corresponding processor using an injection and deliv-

*In SIMD machines such as the Connection Machine, livelock can be addressed by global measures.

ery buffer and with each of its $\log N$ neighbors via an asynchronous channel protocol, using an input and an output buffer per channel, called the Input and Output Frames, capable of holding one message each.

Let us consider a hypercube with N nodes, $D = \log N$ dimensions and nodes A and B, neighbors along dimension i, $0 \leq i < D$. We assume that nodes A and B are connected along dimension i by a bidirectional channel . If node A wants to send a message to B, it places the message in its Output Frame(i). The message can be transmitted if the Input Frame(i) of B is empty and the channel is available in the A to B direction. When in Input Frame(i) of node B, the message will remain there until node B is ready to read from that particular dimension. For the following we assume that a full/empty bit is associated with each Input and Output Frame to implement this interface.

Besides the $\log N$ pairs of Input and Output Frames, each routing node consists of a Queue and a source of randomness that, upon request, selects one of the messages in the Queue with equal probability. The block diagram of a routing node is shown in Figure 1.

Each routing node examines its dimensions in a cyclical order, executing the following infinite loop:

```
current_dim=0;
while (True) do begin

  current_dim=current_dim+1 mod D;
  while(Output Frame(current_dim)==Full)
       {
        current_dim=current_dim+1 mod D;
       }
  Match (current_dim);
  if (Queue==Full AND no match AND
      Input Frame(current_dim)==Full)
      {
       Deroute;
      }
  Send(current_dim);
  Read(current_dim);
end.
```

where *Match*, *Read* and *Send* are defined as follows:

Match examines the messages in the Queue in a FIFO order and selects the first message that can be routed along the current dimension, if such a message exists.

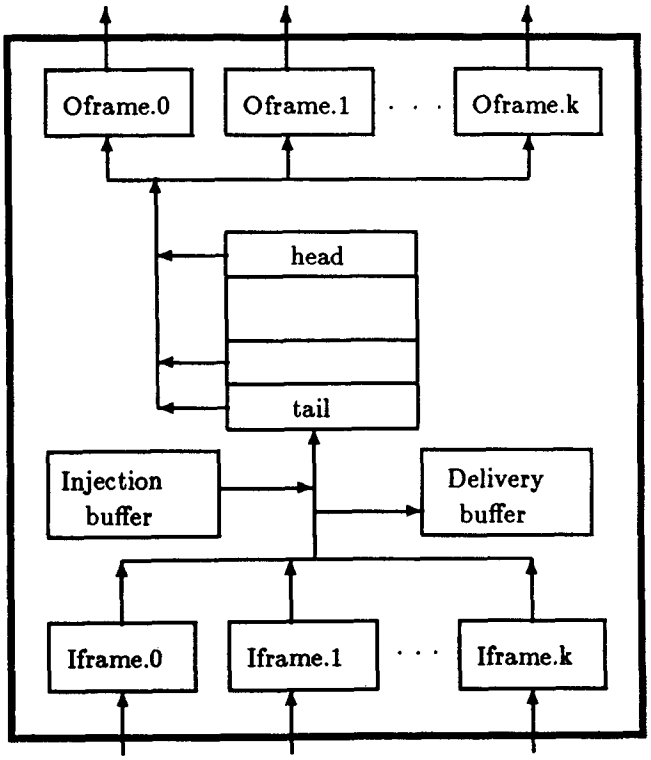

Figure 1: Block diagram of a routing node.

Deroute randomly selects a message to be derouted from the Queue. This operation is invoked if the previous *Match* fails to produce a message to be routed, the Queue is full and a message has to be read from the Input Frame of the current dimension.

Send removes from the Queue the message selected by the previous Match or Deroute operation, if such a message exists, and places it into the Output Frame of the current dimension.

Read removes a message from the Input Frame of the current dimension, if such a message exists. If the message has reached its destination, it is delivered, otherwise it is added to the end of the Queue. If the Queue is not Full and there is a message waiting in the injection buffer, it is added to the end of the Queue. Notice that one might choose to control the injection rate by limiting the frequency of injection or by requiring that new messages are injected iff the Queue capacity is below some predetermined threshold.

Deadlock, livelock and starvation freedom

In every multicomputer network with finite size buffers and continuous, local routing one would like to guarantee the following properties:

- Deadlock freedom - no message is permanently stalled.
- Livelock freedom - every injected message is eventually delivered.
- Starvation freedom - every node is able to inject a message.

In the following, we will examine in detail the chaos router with respect to these three properties. Before proceeding, we should emphasize that we are describing a practical router; all its primitive operations, like a channel transmission, Match, Read etc., take a bounded amount of time to complete. Thus in the following, whenever we mention that an event is of finite duration, this means that there is a maximum amount of time such that the event is guaranteed to complete in time less or equal than the maximum.

Freedom from deadlock. Freedom from deadlock is in general easy to guarantee in nonminimal routers such as the chaos router. As opposed to previously suggested nonminimal routers that guarantee progress at the node level, the chaos router guarantees progress along each dimension of a node. We believe this is a major strength of our router and significantly increases its fault-tolerance.

Let us consider message M that has already been injected into the network and is currently in some node P. Within node P, message M can be in one of three places

- In some Output Frame(i), $0 \leq i < \log N$.
- In some Input Frame(j), $0 \leq j < \log N$.
- In the Queue.

In order to show that it is impossible for a message to be stalled in any of the above cases, we will need the following two lemmas.

Lemma 1: Consider node P and dimension $i, 0 \leq i < \log N$, such that node P last examined dimension i at time t_0. Then, node P will examine dimension i at time $t_0 + \delta$ where δ is finite.

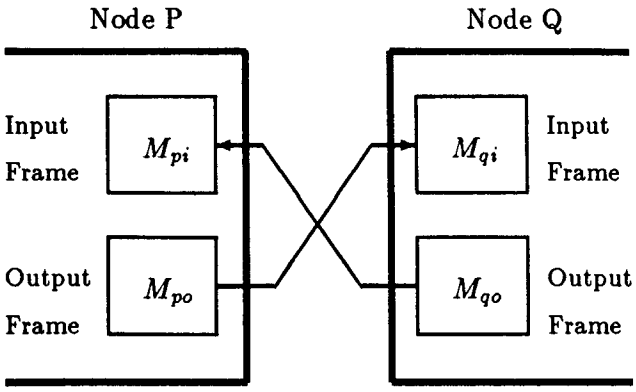

Figure 2: Deadlock configuration.

Proof: By the construction of our algorithm, node P executes an infinite loop, examining its dimensions in a cyclical order. For each dimension, if the Output Frame of that dimension is full the node proceeds to examine the next dimension, otherwise it performs one or more of the following operations: Match, Deroute, Send and Read. The duration of each one of these operations, although not constant, depends only on the state of node i; there is no need for communication or synchronization between neighboring nodes in order for these operations to complete. Thus, it takes a finite amount of time for a node to service each dimension and as a result, the elapsed time δ between two subsequent examinations of some dimension i is finite.

Lemma 2: Consider nodes P and Q, neighbors along dimension $i, 0 \leq i < \log N$, such that neither P nor Q is currently examining dimension i. Then one of the four I/O Frames (the Input and Output Frame(i) of P and Input and Output Frame(i) of Q) must be empty.

Proof: For the sake of contradiction let us assume that all four I/O Frames are full. Let M_{pi} be the message in the Input Frame(i) of P, M_{po} the message in the Output Frame(i) of P, M_{qi} the message in the Input Frame(i) of Q, and M_{qo} the message in the Output Frame(i) of Q. This configuration is shown in Figure 2. We will show that this configuration is impossible.

Let us consider the order in which the messages were placed in the I/O Frames. Message M_{pi} was placed in the Input Frame of node P after message M_{po} was placed in the Output Frame of node P. Otherwise, by the construction of the algorithm,

M_{pi} would have been read following the Send operation that placed message M_{po} in the Output Frame(i) of P. Let us denote this ordering of events as $M_{po} < M_{pi}$ ($\Sigma 1$). Similarly, message M_{qi} was placed in the Input Frame(i) of node Q after message M_{qo} was placed in the Output Frame(i) of node Q. Let us denote this ordering of events as $M_{qo} < M_{qi}$ ($\Sigma 2$).

Furthermore, message M_{pi} was placed in the Input Frame(i) of node P before message M_{qo} was placed at the Output Frame(i) of Q. This is true because M_{qo} moved to the Output Frame(i) of Q as soon as this Frame became empty, an event that happened when M_{pi} moved from the Output frame(i) of Q to the Input Frame(i) of P. Thus $M_{pi} < M_{qo}$ ($\Sigma 3$).

Similarly, message M_{qi} was placed in the Input Frame(i) of node Q before message M_{po} was placed in the Output Frame(i) of P, thus $M_{qi} < M_{po}$ ($\Sigma 4$).

By combining inequalities ($\Sigma 1$),($\Sigma 2$),($\Sigma 3$) and ($\Sigma 4$) we have $M_{po} < M_{pi} < M_{qo} < M_{qi} < M_{po}$ which is a contradiction. ■

We should note here, that while a node is examining dimension i, it is possible for all four I/O Frames to be full, but only momentarily. This can happen between a Send and a Read operation. It is easy to see, that the Read operation will result in an empty Frame.

Now we are ready to show freedom from deadlock.

Theorem 1: For every message M that enters Output Frame(i), $0 \leq i < \log N$ of some node P at time t_0, M is guaranteed to leave the Output Frame at time $t_0 + \delta$ where δ is finite.

Proof: Consider nodes P and Q that are neighbors along dimension $i, 0 \leq i < \log N$ and message M that node P placed in its Output Frame(i) at time t_0. By the construction of our router, the only possible move for message M is that from the Output Frame(i) of P to the Input Frame(i) of Q. Thus, there are two cases, either the Input Frame(i) of Q is full or empty:

1. The Input Frame(i) of node Q is full. Then there are two cases.

   a. The Output Frame(i) of Q is full. Then from Lemma 2 the Input Frame(i) of node P must be empty. Thus, the Output Frame(i) of Q can transmit its message to the Input Frame(i) of node P as soon as the channel is transmitting in the P to Q direction. Assuming finite length messages, transmission is guaranteed to complete in a finite amount of time, leaving the Output Frame(i) of Q empty. Then, the state of the system is described by the following case 1b.

   b. The Output Frame(i) of Q is empty. Then, the first time that node Q examines dimension i, an event that is guaranteed by Lemma 1, it will read from its Input Frame(i). Thus, Input Frame(i) of Q is guaranteed to become empty within a finite amount of time, resulting in a system state as described in the following case 2.

2. The Input Frame(i) of node Q is empty. In this case, the Output Frame(i) of P can transmit its message to the Input Frame(i) of Q as soon as the channel is transmitting in the P to Q direction, an event which is guaranteed to happen within a finite amount of time.

Theorem 2: For every message M that enters Input Frame(i), $0 \leq i < \log N$ of some node P at time t_0, M is guaranteed to leave the Input Frame at time $t_0 + \delta$ where δ is finite.

Proof: By the construction of our router, there are two possibilities:

1. The Output Frame(i) of P is full. As long as this condition persists, each time node P examines dimension i it will proceed to the next dimension. But by Theorem 1 we know that Output Frame(i) is guaranteed to become empty in a finite amount of time, resulting in a system state as described in the following case 2.

2. The Output Frame(i) of P is empty. Then, the first time node P examines dimension i, it will perform a Read operation on the Input Frame(i). If message M has reached its destination it will be placed in the delivery buffer, and we assume that this can happen in finite time, otherwise it will be added at the end of the Queue. Node P is guaranteed to have an empty slot in its Queue for message M since it can deroute a message along the empty Output Frame(i).

Theorem 3: For every message M that enters the queue of some node P, M is guaranteed to leave the queue in a finite amount of time.

Proof: Let us consider message M that enters the queue of some node P at time t_0; let us assume that M is the i-th message in the queue, where $1 \leq i \leq c$ and c is the queue capacity. Clearly, M can be routed along at least one dimension; let such a dimension be j. Furthermore, there are at most $i-1$ messages in the queue ahead of M that can be routed along dimension j. Let m be the number of such messages.

Let $t_1, t_2, ...t_m, t_{m+1}$ be the times that node P sends a message along dimension j, where $t_{i+1} = t_i + \delta_i, 0 \leq i \leq m$.

If right before time t_{m+1} message M is still in the queue, it is the first message in the Queue that can be routed along dimension j and it will be routed at time t_{m+1}.

By Theorem 1 and Lemma 1, intervals $\delta_i, 0 \leq i \leq m$ are of finite length and message M is guaranteed to leave node P in a finite amount of time.

Livelock arguments. Having established freedom from deadlock, let us now consider freedom from livelock. In nonminimal routers, as messages are allowed to move further from their destinations, some mechanism is required in order to guarantee that messages will eventually make progress towards their destinations. We have argued that the mechanism to *deterministically* guarantee message delivery, *i.e.* priority routing, is quite expensive. In the following we will show that by choosing randomly a message to deroute, the chaos router guarantees message delivery with high probability. Lemma 3, formalizes the key idea of our router: no message is ever derouted with certainty or in other words, every message has a nonzero chance of avoiding derouting.

Lemma 3: $\exists \epsilon > 0$, such that for every message M entering the queue of some node P , M will be routed with probability $p \geq \epsilon$ and derouted with probability $q = 1 - p$.

Proof: By Theorem 3, there is a maximum amount of time $T = \sum_{i=0}^{m} \delta_i$ that message M can stay in the queue of some node P after which M will certainly be routed. During this time T, node P makes a finite number of routing decisions. Thus message M can be subjected to at most a finite number, r, of derouting decisions. At each derouting decision, a message is randomly selected from the queue with uniform probability. The probability that message M can escape derouting at each derouting decision is $\frac{c-1}{c}$. Thus, the probability that message M will be routed is at least $\epsilon = (\frac{c-1}{c})^r$.

Main result. Using Lemma 3, we will prove in the following that the probability of long paths in the network diminishes as their length increases.

Let us define the path of a message in the network as a sequence of *moves*. At each move, message M either moves closer to its destination with probability $p \geq \epsilon$ or further from its destination with probability $q = 1 - p$. Clearly, a message cannot move further than $\log N$ from its destination.

Let us define a *game* as a sequence of $\log N$ moves. Message M starts game i at distance a_i and finishes at distance a_{i+1}. Let l_i denote the event that M was not delivered during game i and w_i the event that M was delivered during game i.

Let $Q(i)$ be the probability that message M has not been delivered after i games. Then

$$Q(i) = P(l_i l_{i-1}..l_1) = P(l_i \mid l_{i-1}..l_1) \cdot P(l_{i-1}..l_1)$$

For simplicity, let us substitute F_k for $l_k...l_2 l_1$, $1 \leq k < i$ and let us define $P(l_1 \mid F_0) \equiv P(l_1)$ and $P(w_1 \mid F_0) \equiv P(w_1)$. Then

$$Q(i) = P(l_i \mid F_{i-1}) \cdot P(l_{i-1} \mid F_{i-2}) \cdots P(l_1) \quad (1)$$

Clearly, $P(l_j \mid F_{j-1}) = 1 - P(w_j \mid F_{j-1})$, $1 \leq j \leq i$. In the following we will estimate $P(w_j \mid F_{j-1})$. Let $S_{j,k}$ denote the event that message M starts game j at Hamming distance k from its destination. Events $S_{j,k}$ are mutually exclusive and one of them necessarily happens. Thus

$$P(w_j \mid F_{j-1}) = P(w_j S_{j,1} \cup .. \cup w_j S_{j,\log N} \mid F_{j-1}) \Rightarrow$$

$$P(w_j \mid F_{j-1}) = \sum_{k=1}^{\log N} P(w_j S_{j,k} \mid F_{j-1}) \Rightarrow$$

$$P(w_j \mid F_{j-1}) = \sum_{k=1}^{\log N} P(w_j \mid S_{j,k} F_{j-1}) \cdot P(S_{j,k} \mid F_{j-1}).$$

But $P(w_j \mid S_{j,k} F_{j-1}) \geq \epsilon^{\log N}$, thus

$$P(w_j \mid F_{j-1}) \geq \epsilon^{\log N} \sum_{k=1}^{\log N} P(S_{j,k} \mid F_{j-1}).$$

Since $\sum_{k=1}^{\log N} P(S_{j,k} \mid F_{j-1}) = 1 \Rightarrow$

$$P(w_j \mid F_{j-1}) \geq \epsilon^{\log N} \Rightarrow$$

$$P(l_j \mid F_{j-1}) \leq 1 - \epsilon^{\log N} \quad (2)$$

Finally, (1),(2) $\Rightarrow Q(i) \leq (1 - \epsilon^{\log N})^i$.

Thus the probability that M will not have been delivered after i games, where $i \rightarrow \infty$ is:

$$\lim_{i \rightarrow \infty} Q(i) = (1 - \epsilon^{\log N})^i = 0$$

The probability $P(i)$ that M will be delivered after i games, where $i \rightarrow \infty$ is:

$$\lim_{i \rightarrow \infty} P(i) = 1.$$

Starvation arguments. As described previously, nodes are allowed to inject messages in the network iff the Read operation finds the Queue not full. Consider the following scenario. A node starts with a full queue and every time it services some dimension, it has to read a message that has not reached its destination yet. Such a node will be prevented from injecting. Different solutions have been suggested that address this problem.

The first solution is based on the observation that any message delivered to a node, gives the node the right to inject a new message into the network. It requires that for every message delivered by the network, a Return message is sent back to the original sender [Ngai and Seitz 89]. Clearly, this method increases the network traffic dramatically, flooding the network with messages that contain no information.

A different solution [Ngai and Seitz 89] requires that nodes notify their neighbors each time they inject a message. If a node's injection rate is higher than the injection rate of its neighbors, the node has to stop injecting. This method is also expensive, requiring record-keeping and dedicated pins per channel for nodes to exchange injection rate information.

In the following we present a new, simple and efficient injection policy. Our method is based on the concept of an *injection token* that visits all nodes in some predetermined order giving them the right to inject.

Let us consider a hypercube with N nodes and a directed Hamiltonian cycle containing the N nodes. For each node i, $0 \leq i < N$, let us define $Neighbor(i) = j$, with $0 \leq j < N$ such that (i,j) is an edge of the Hamiltonian cycle. Furthermore, let us consider a message marked in some unique way, called the *injection token*. When node i receives the injection token, it determines whether it has been prevented from injection longer than some predetermined time T_d. If not, it sends the injection token to node j, where $Neighbor(i) = j$. Otherwise, it consumes the injection token thus creating one empty slot in the queue which can be filled by a newly injected message. A message injected into the network following the consumption of the injection token has to be marked appropriately; let us call such a message a *tagged* message. Upon delivery of a tagged message, the injection token has to be regenerated. Then there are two cases:

1. Nodes can compute the address of any Neighbor in the Hamiltonian cycle. Then, the recipient of a tagged message can regenerate the injection token and send it directly to the Neighbor of the sender of the tagged message.

2. Nodes are assigned rather than compute the address of their Neighbors. In this case the recipient of a tagged message has to send a *null* message to the sender in order to give it the chance to regenerate the injection token. The transmission of this null message can be avoided iff every node knows the Neighbor of every other node in the network. The advantage of assigning rather than computing Neighbors is fault-tolerance. In the presence of one or more faulty nodes, a new Hamiltonian cycle can be computed that avoids the faulty nodes. Then a new assignment of neighbors can take place.

Clearly, our injection policy guarantees access to the network subject to guaranteed delivery of the three marked messages required to implement the protocol. Thus, our injection policy guarantees access to the network in a probabilistic way. Finally, we should mention that more than one token can be used, each one visiting either all nodes as described previously or different subsets of the nodes.

Performance evaluation

The key issues for any practical router are implementation efficiency and performance. In the previous sections we presented the novel livelock and starvation prevention schemes of the chaos router.

Both these schemes can be efficiently implemented and greatly simplify the node's logic. In this section we will address the performance issue. We will present performance measurements of the chaos router and compare it with the performance of two other routers of similar nature: the *priority* router and the *natural* router.

The priority router, differs from the chaos router in that it assumes *age* information for every message, assigns higher priorities to older messages and routes messages according to their priority in order to guarantee message delivery. In order to implement this age-priority scheme, a priority queue is used in each node. Although the priority router has to pay the overhead of comparing the ages of messages, it does have one performance advantage over the chaos router. In particular, in the chaos router messages are serviced in a FIFO order; thus older messages can "loose" to younger messages. In order for the chaos router to outperform the priority router, the overhead of age comparisons of the priority router should outweigh the possible "out of order" delays of the chaos router.

The natural router, is almost identical to the chaos router except for the way it deroutes messages. The natural router always deroutes the last message in the Queue. The performance of the natural router is of interest, as it allows us to evaluate the relative performance effect of randomization in the chaos router.

We measured performance using an event-based simulator of a 256 node $n$cube. More than 150,000 messages (at least 600/node) were injected in each simulation run. The routers were specified at a fine grain. All three routers were assumed to have the same basic structure and execute similar Match, Read, Send and Deroute operations. Time was measured in units of a "step", which is defined to be the time for a router to decide if a message can be routed along dimension k; this is likely to be a few gate delays.

The routers were compared assuming both fast channels (5 steps per transmission) and slow channels (100 steps per transmission). Thus, for the same message size, fast channels can be interpreted as "wide" or high bandwidth channels and slow channels can be interpreted as "narrow" or low bandwidth channels. Similarly, for the same channel bandwidth, the performance of fast and slow channels is an indicator of the effect of message size. Tables 1a-3a show the results under fast channels and tables 1b-3b the results under slow channels. In each category (fast, slow) the same load was applied to all routers, that load being the maximum sustained load by the slowest router.

Three types of traffic patterns were used:

- Destinations selected uniform randomly (Tables 1a and 1b).
- A randomly selected set of nodes 3 times more likely to be destinations than other nodes (Table 2a and 2b).
- *Transpose* pattern, *i.e.* the destination is the source address bit-sequence with the front and back halves exchanged (Table 3a and 3b).

The first case is the standard (and probably unlikely) random traffic. The second corresponds to a mild form of "hot spots." The third case is designed to incur substantial congestion in the "middle" of the message routes.

The quantities reported are the average message delay in steps, the worst observed delay of any message in steps, the maximum number of times any message is derouted and the fraction of derouting moves versus total moves.

The available results show the chaos router to be at least 20% and as much as 70% faster than the priority scheme. Thus, the latency of performing comparisons in a priority router far outweighs the benefits of "in order" delivery.

Conclusions

We have presented the chaos router, an adaptive, nonminimal, randomized router. By randomly selecting which message to deroute, the chaos router eliminates the need for priority routing. This greatly simplifies the router's logic. More importantly, while priority-based routers have to prioritize messages all the time, in the chaos router the overhead of the livelock prevention mechanism is introduced only when a message has to be derouted. We also presented an efficient scheme for starvation prevention that significantly reduces the communication requirements of previously suggested schemes. Simulation results exhibit the good performance of the chaos router under random traffic and traffic known to create hot-spots.

Fast channels

Router	Average delay	Worst delay	Max Drt	Deroute fraction
priority	733.85	6180	2	$7.6\ 10^{-4}$
chaos	592.68	1563	2	$7.8\ 10^{-5}$
natural	544.58	1558	1	$4.9\ 10^{-5}$

Table 1a: Performance measurements under uniform random traffic.

Slow channels

Router	Average delay	Worst delay	Max Drt	Deroute fraction
priority	2179.42	11078	2	0.011
chaos	1885.42	5191	2	$1.5\ 10^{-3}$
natural	1809.86	4559	2	$1.1\ 10^{-3}$

Table 1b: Performance measurements under uniform random traffic.

Router	Average delay	Worst delay	Max Drt	Deroute fraction
priority	790.99	8016	2	$3.8\ 10^{-3}$
chaos	626.59	1943	2	$2.6\ 10^{-4}$
natural	570.25	1690	2	$2.6\ 10^{-4}$

Table 2a: Performance measurements for 3X hot-spots.

Router	Average delay	Worst delay	Max Drt	Deroute fraction
priority	2250.18	13492	3	0.019
chaos	1892.69	6035	3	$3.5\ 10^{-3}$
natural	1820.92	5200	2	$3.0\ 10^{-3}$

Table 2b: Performance measurements for 3X hot-spots.

Router	Average delay	Worst delay	Max Drt	Deroute fraction
priority	1597.57	6039	3	$5\ 10^{-2}$
chaos	894.60	2389	2	$1.9\ 10^{-3}$
natural	911.82	2127	2	$2.2\ 10^{-3}$

Table 3a: Performance measurements for "Transpose."

Router	Average delay	Worst delay	Max Drt	Deroute fraction
priority	3432.93	13540	3	0.06
chaos	2425.49	7346	4	$7.5\ 10^{-3}$
natural	2362.90	5951	2	$5.7\ 10^{-3}$

Table 3b: Performance measurements for "Transpose."

Variants of chaotic routing, currently under investigation include a router which, when in the derouting state, selects the last message in the queue and deroutes it along a randomly chosen dimension, and a router where no randomization is introduced but the randomness of the system is exploited to prevent livelock.

Acknowledgements

The authors thank R. Cypher for his valuable comments and help with the probabilistic proof, and C. Ebeling for many challenging discussions.

References

[1] A. Borodin and J.E. Hopcroft. Routing, Merging and Sorting on Parallel Models of Computation. *Journal of Computer and System Sciences*, vol. 30, pp. 130-145, 1985.

[2] J.N. Ngai and C.L. Seitz. A Framework for Adaptive Routing in Multicomputer Networks. *Proc. of the 1989 ACM Symposium of Parallel Algorithms and Architectures*, pp.1-9.

[3] W.J. Dally and P. Saung. Design of a Self-Timed VLSI Multicomputer Communication Controller. *Proceedings, International Conference on Computer Design*, 1987, pp. 230-234.

[4] C. Kaklamanis, D. Krizanc and A. Tsantilas. Tight Bounds for Oblivious Routing in the Hypercube. *Proc. of the 1990 ACM Symposium of Parallel Algorithms and Architectures*, (this issue).

[5] L.G. Valiant and G.J. Brebner. Universal Schemes for Parallel Communication. *Proceedings of 13th Symposium on Theory of Computing*, ACM, pp. 263-277, 1981

[6] Smaragda Konstantinidou. Adaptive, Minimal Routing in Hypercubes. *6th MIT Conference on Advanced Research in VLSI*, 1990, pp. 139-153.

[7] W. Daniel Hillis. *The Connection Machine.* MIT Press, 1985

[8] D.A. Padua, D.J. Kuck and D.H. Lawrie. High-Speed Multiprocessors and Compilation Techniques. *IEEE Transactions on Computers*, Vol. C-29, pp. 763-776, September 1980.

[9] D. Gelernter. A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks. *IEEE Transactions on Computers*, vol. C-30, pp. 709-715, Oct. 1981.

[10] D.C. Grunwald. *Circuit Switched Multicomputers and Heuristic Load Placement.* Ph.D. thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, September 1989.

[11] P.C. Yew. *On the Design of Interconnection Networks for Parallel and Multiprocessor Systems.* Ph.D. thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, March 1981.

[12] P.Y. Chen. *Multiprocessor Systems: Interconnection Networks, Memory Hierarchy, Modeling and Simulations.* Ph.D. thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, January 1982.