Randomized Routing For Message Passing Using Virtual Channels: The "Inebriated Router" Algorithm[1]

Sramana Mitra and Dominique Thiébaut
Department of Computer Science, Smith College, Northampton, MA 01063

Abstract. A randomized routing algorithm for message passing on hypercube networks is presented, along with its application to transputer networks. The performance of the algorithm is compared to other previously published models, including the shortest path algorithm, Valiant's two-phase randomized algorithm, and Greenberg and Hajek's deflection algorithm. The results of a comparative study concentrating on *mean packet delay*, *queue length*, *node population*, and the *distribution of deflection of a packet from its shortest path* are given. The new algorithm is shown to perform better than its counterparts when packet destinations exhibit hot spots.

1. Introduction

In this paper, we propose a new randomized routing algorithm for hypercube networks and analyze its performance on a transputer hypercube. Earlier published randomized routing algorithms for hypercubes outperform direct routing ones by adopting a two-phase approach. First, packets are sent to randomly selected intermediary destinations. Then the packets are sent from these scattered nodes to their original destination. The randomness introduced in the first phase reduces hot-spots and the resulting contention at relay nodes, yielding improved performance measures, such as mean packet delay, or mean queue length. Our algorithm differs from these approaches in that the two phases are merged, and the randomness is distributed over time.

When a packet is generated by a source and sent towards its destination, it accomplishes its travel by hopping from relay node to relay node, until it reaches the destination node. Our algorithm divides the transmission links leaving each relay node into two groups. The first group, *Group I*, contains the links that offer the *shortest path* to the destination of the packet. The second group, *Group II,* contains the links that do not[2]. The relay nodes introduce randomness in the routing in two ways.

1. Each relay node conducts a Bernoulli trial to select one of the two groups. With a (high) probability p used homogeneously throughout the network, the relay node picks Group I. With a much lower probability $(1-p)$, Group II is chosen.
2. Once a group is selected, one of its links is selected at random, and the packet is sent on that link. Thus, two levels of randomness control the non deterministic routing of the packets throughout the network.

---

[1]Accepted for presentation at the North American Transputer User Group (NATUG) meeting, May 1993, Vancouver, British Columbia.

[2]Group II may be empty if the destination is antipodal to the sending node

As the title indicates, this method of routing is similar to having an *inebriated router* which, when faced with several possible choices for routing a message, would always pick one at random, and every so often, would send a packet on the wrong path, sometimes even back to the node from which the packet just came. As a result, routed packets travel through two different kinds of random paths. The first kind can be illustrated by setting p to 1. In this case two packets issued from the same source and traveling toward the same destination may follow different shortest paths. The length of the path is identical for both packets, but the nodes visited on the way may be different. The reason is that although Group I is chosen with probability 1, the link selected to carry the packet is chosen at random if more than one such link are available. On the other hand, when p is lower than 1, it is possible for some relay nodes on the path to route the packet in the wrong direction by choosing a link from Group II. This time two packets with identical sources and destinations may follow paths of different lengths.

In this paper we explore the influence of the mapping of high dimensional hypercubes onto smaller ones on three performance measures (mean packet delay, queue size and node population) and we show that when the nodes are flooding the network with packets having a few allowed destinations, our algorithm produces mean packet delays that are shorter than those of the other randomized routing algorithms.

2. Background

The two-phase random algorithm was first introduced by Valiant [8], and applied to various other networks. Mitra and Cieslak [6], along with Leighton *et. al.* [4] represent such efforts. Greenberg and Hajek [3] use an approach similar to ours in a different context. In their *deflection routing* approach, relay nodes with output buffers that are full deflect packets by sending them on a randomly selected "empty" link. Their randomness is controlled by the status of the buffers at the relay nodes, while ours is not. To our knowledge, this paper is the first to provide measured performance for the deflection algorithm. Compared to two-phase algorithms, the main advantage of such a method is that it does not require a lengthy first phase of packet scattering, which, although it can be made to add only a constant number of parallel steps, introduces significant overhead. The disadvantage is that reordering delays will be introduced at the destinations where packets may arrive out of sequence, because consecutive packets follow different paths. However, such delays (very likely significant) can easily be accounted for by using estimates such as the ones presented by Baccelli [2].

Peleg and Upfal have studied *oblivious routing* algorithms [7] in which the route of one packet is not influenced by the source or destination of any other packet in the system. Our algorithm differs from theirs in that we choose the next node to receive a packet. Consequently, the path is independent of the other packets, even though it is influenced by the destination of the packet itself. Aiello [1] describes algorithms for bit-serial routing on hypercubes, but none of the algorithms are oblivious. Aiello's approach incorporates some degree of randomization in that each packet is first routed to a random intermediate node, and then routed to its true destination, using Valiant's original paradigm discussed earlier.

3. Virtual Channels and The Experiments

To measure the performance of our algorithm, we conducted experiments on a 32-transputer Superset system manufactured by CSA (Computer Systems Architects), and Version 91.1 of Logical System's C compiler is used to code the different test programs.

The recent release of Logical Systems' C compiler supports *virtual channels*, making it appealing to implement hypercubes of degrees larger than 4 (the current limit with 4-channel transputers) and preparing the ground for the T9000, which will support virtual channels in VLSI [5]. Hypercubes of degrees larger than 4 can be mapped to 4- or lower dimension hypercubes by collapsing the cubes formed in higher dimensions onto single transputers, and by using virtual channels to connect the logical nodes mapped to different transputers. Figure 1 shows a 3-D hypercube mapped onto a 2-D hypercube. The connection between Node T0 and T1, for example, uses a virtual channel sharing the same hardware link as the virtual channel connecting Nodes T4 and T5. The connection between T0 and T4, on the other hand is via a virtual channel that is memory mapped, since both nodes are implemented on the same physical transputer. This feature makes it ideal to test the behavior of routing algorithms such as the ones we are considering in hypercubes of varying size.

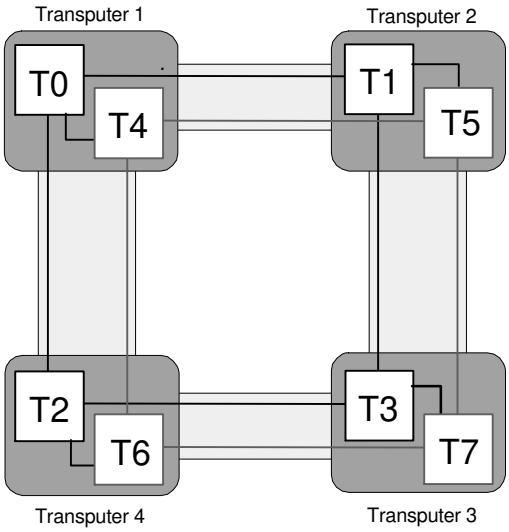

Figure 1: A 3-dimensional hypercube mapped to a 2-dimensional hypercube. Each physical transputer supports the code for two logical nodes. Each hardware link supports two virtual channels.

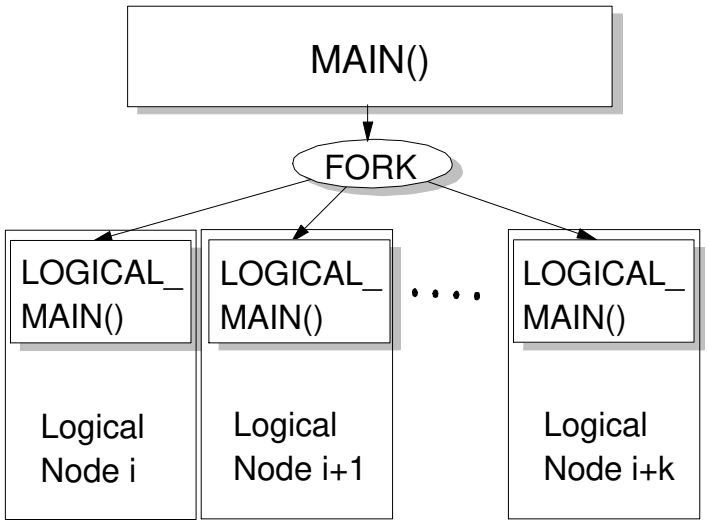

Figure 2: Software mapping. Main procedure runs logical_main() procedures in parallel, one for each logical node mapped to the transputer.

3.1 Logical to Physical Mapping

In this section we briefly describe our software implementation. In our description of the implementation of the different parallel tasks, we shall use the C terminology that the first procedure that starts the program is called *main()*. Following this convention, Figure 2 shows how the main program of a particular transputer calls in parallel the *logical_main()* programs representing the logical nodes that are mapped to that transputer. If there are *PMAX* physical transputers, and *LMAX* logical nodes mapped to them, then each physical node runs *LMAX/PMAX* logical_main task(s) in parallel. Each logical_main, in turn, starts up a *router* in charge of routing packets on the network, and several *input* and *output managers* to control the reception and transmission of packets on the virtual channels.

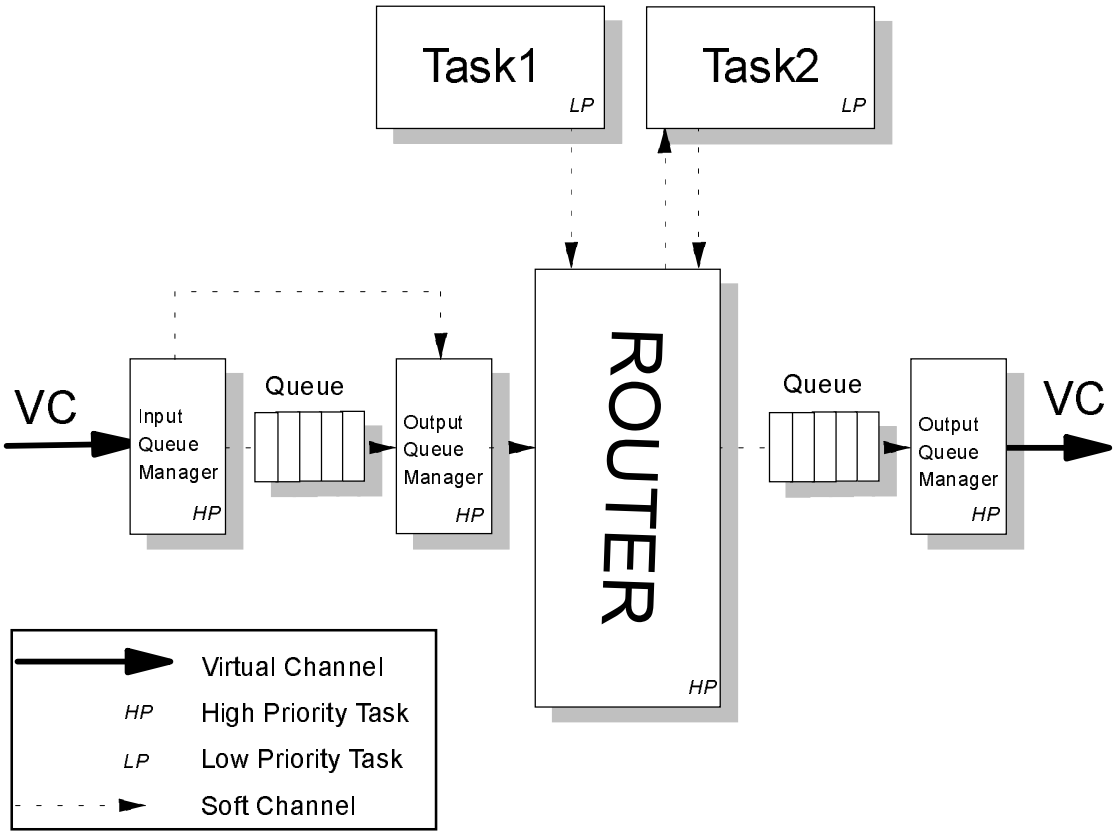

Figure 3: Block diagram of software controlling a logical node.

These three tasks are run at high priority. The logical_main procedure also starts *Task1* and *Task2* (as low priority tasks), which control the generation of new packets, the reception of messages, and the recording of the statistics. Figure 3 shows a block diagram of these different tasks.

3.2 Synchronization: Phase 1 of The Experiment

Running the experiments requires 3 phases: *i*) Synchronization, *ii*) Routing, and *iii*) Measurement .

In the first phase, all the transputers must be synchronized so that accurate recording of packet delays be possible. This step is important since the measuring of the delay experienced by a packet going from *A* to *B* requires that both *A* and *B* tag the packet with a time stamp. Hence, both *A* and *B* should maintain the same time.

The synchronization is initiated by the Logical Node 0, which first sets its own high priority clock to T_0, and then sends a series of 20 *ping-pong* packets to each node in the network. When a node receives such a packet, it automatically returns it to its sender. This way, Logical Node 0 can measure the average round trip delay, and compute an accurate estimate of the time it takes a packet to reach a node. After the 20th ping-pong packet is sent and received, a *Set-Time* packet is then sent, containing the time at which the packet left Logical Node 0, plus the estimated travel time to the destination. Upon receiving this packet, the destination sets its own (high priority) timer to the time value included in the packet just received.

3.3 Phase 2: Routing

Once the synchronization is over, the experiment moves on to the *routing* phase, where several packets are sent to random destinations, flooding the network. In each node Task1 sends a predetermined number of packets to random destinations, one after the other. These packets are tagged as *Info* packets, and are used to measure packet delays, the size of queues entered along the path, and the population of nodes visited.

Although the destinations are chosen at random, the user can control the degree of *hotness* of the network nodes. We define the hotness as the number of nodes that are allowed to receive packets. A hotness of 1 indicates that only one node in the network will receive packets. This node is referred to as a *hot node*. A hotness of *LMAX*, on the other hand, indicates that each node in the network is a potential target. The need for sophisticated routing algorithms stems from the creation of hot spots in networks resulting from the data exchange required by the running application, and it is necessary to create artificial hot spots to rate the effectiveness of the routing mechanism. A hotness of 1 yields the worst case conditions for a routing algorithm, while a hotness of *LMAX* (uniform distribution) does not present the routing algorithm with any challenge.

In our implementation, the router task communicates with input and output managers via soft channels, and is in charge of implementing the details of four different routing algorithms, which we will refer to as the Inebriated Router, Valiant, Greenberg, and Shortest Path algorithms.

All input and output virtual channels are buffered and are serviced by the input and output managers, all high priority tasks. The managers communicate with the router via soft channels.

3.3.3 Measurement Conventions

All the data required for our analysis are gathered from the information stored in the *Info* packets. The *Info* packet has fields for recording *source*, *destination*, *time sent*, *time received*, *number of hops*, *queue sizes* and *node populations* at the different nodes visited while traveling from source to destination. We refer to these intermediary nodes as *hops*.

In our implementation of the four algorithms we have adopted the following conventions: Queue sizes are recorded as soon as a packet enters a queue, and node populations as soon as a packet enters a node. The number of hops maintained in each packet is incremented by the router of each node, including the destination node. The *Deflection* of a packet is defined as the difference between the actual number of hops taken by a packet and the number of hops corresponding to the shortest path.

During all experiments, the packets are sent one after the other, as fast as the network can absorb them. We refer to this method as *flood filling*, or *flooding* the network.

4 Results

4.1 Effects of Folding High-Dimension Hypercubes Onto Lower-Dimension Ones on the Mean Packet Delays

When several logical nodes are mapped on to one physical transputer, measured statistics may or may not represent the true statistics that would be measured in the unfolded network, and several different mechanisms alter these measured statistics. It is thus interesting to see how the statistics change when high dimension networks are folded onto smaller ones, and to see if prediction of the true performance measures is still possible.

The first element that influences the measures is that each transputer is running the code for several logical nodes in parallel. So the computation time available for each logical node decreases with the number of logical nodes per physical transputer.

On the other hand, we can expect communication between logical nodes mapped to the same transputer to be faster, since in this case the channels are memory mapped, and the transfer of messages becomes a memory copy operation which is faster than message passing over a physical link.

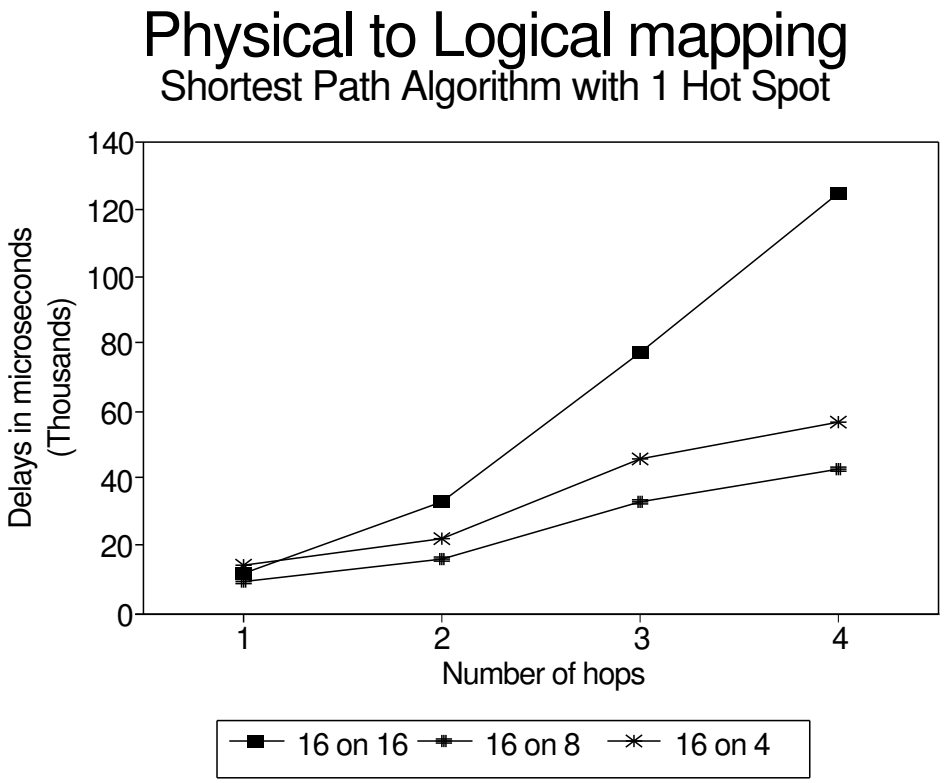

Figure 4: Average packet delay incurred by Shortest Path algorithm controlling the routing of messages flooding a 4-dimensional hypercube mapped onto a 4-D, 3-D and 2-D physical hypercubes.

Figure 4 shows the variation of the average packet delays for the Shortest Path algorithm as a function of the number of hops visited by the packets, and as a function of the logical/physical node mapping: 16 logical nodes mapped onto a 16-node hypercube, onto an 8-node hypercube, and onto a 4-node hypercube. This figure shows the 1 hot spot case, where all packets are sent to one destination only. The Shortest Path algorithm is the only one shown here, as the other algorithms exhibit similar behavior. We observe that the delays for the 16-on-16 mapping are the largest, and that those for the 16 on 8 case are the

smallest, with the 16 on 4 case in-between. We see here the varying effects of the mapping as indicated earlier. The 16 on 8 case presents shorter paths for the packets, while not stressing the transputers much. For the 16 on 4 case, although most paths require on-chip travel, the transputers must run the code for 4 logical nodes in parallel, slowing down the processing of each logical node, and yielding longer delays than for the 16 on 8 case. We also observed (data not shown here for the sake of brevity) that when the packets are distributed uniformly (hotness of *LMAX*) the delay curves for the same network mapped onto different smaller physical nets are parallel to each other. Clearly this area is very rich in research potential, and requires a more thorough investigation which is currently beyond the scope of this paper.

4.2 Effect of Folding on Queue Size

Figure 5 shows the histograms of the variation of the queue size seen by packets as they enter nodes. The first observation is that the queue size seen by incoming packets is relatively independent of the mapping of the logical nodes to the physical ones. This is an interesting property that we observed throughout our experiments.

Of the four algorithms, the Shortest Path is the one which creates larger queues since it makes no efforts to avoid possible hot spots. We see that the other three algorithms behave similarly regarding queue size, with a non surprising edge for Greenberg and Hajek's algorithm which is designed to avoid non empty queues.

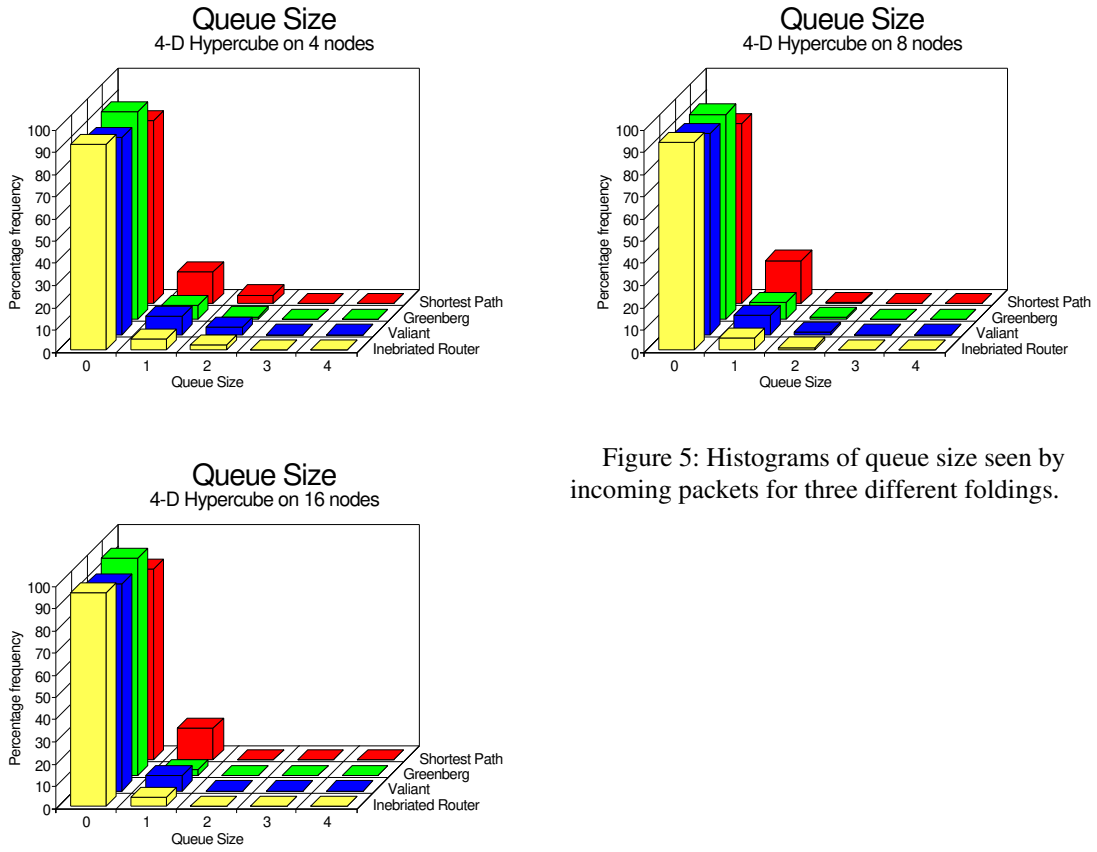

Figure 5: Histograms of queue size seen by incoming packets for three different foldings.

4.3 Effect of Folding on Node Population

The node population represents the total number of messages present in a node when an incoming packet arrives at that node, including itself. Figure 6 shows the histograms for the node population for the four algorithms, under the same three mappings used in Figure 5.

Here again we observe that the histograms show almost consistently the same trends when algorithms are compared to each other and across populations. The exception lies in the scale of the population encountered, with maximums of 9, 6 and 5, for the 16-on-4, 16-on-8, and 16-on-16 mappings, respectively.

In terms of absolute performance, we observe that the Inebriated Router algorithm performs best for total node storage, while Valiant and Greenberg maintain the highest node population over the three mappings.

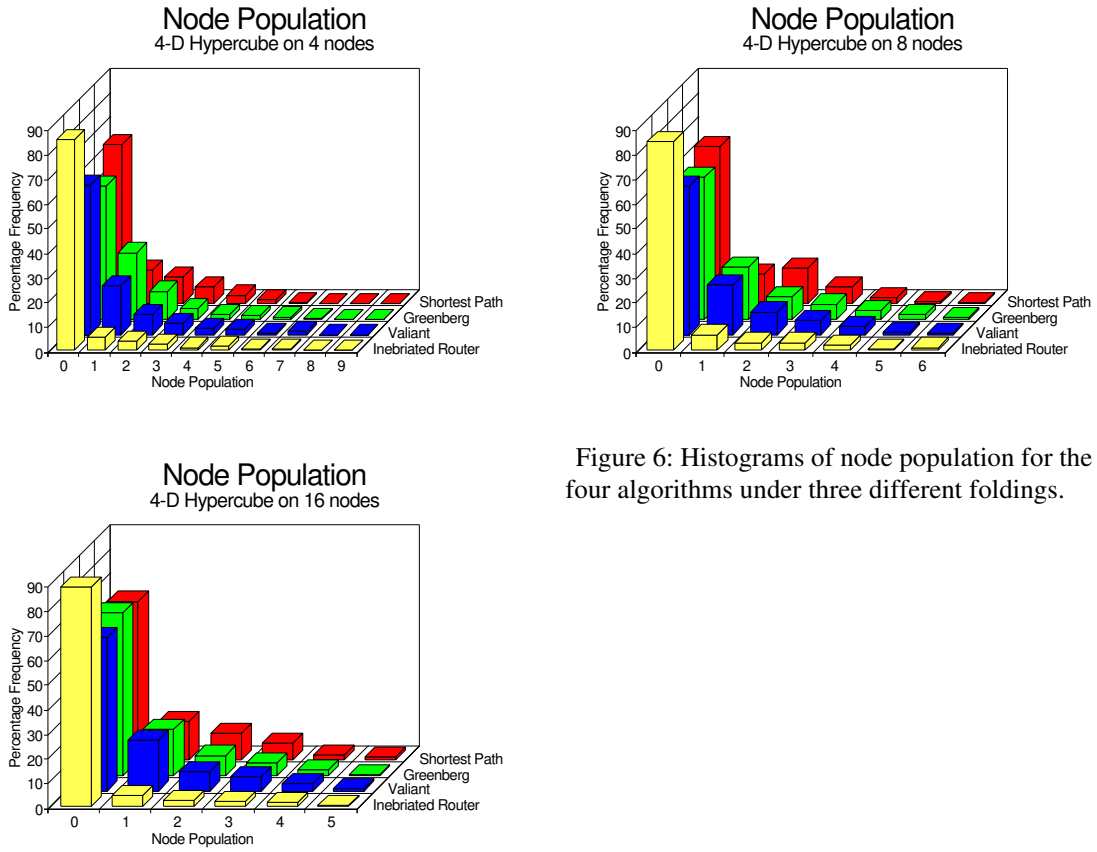

Figure 6: Histograms of node population for the four algorithms under three different foldings.

4.4 Tuning The Performance of the Inebriated Router Algorithm

The performance of the Inebriated Router algorithm depends on the probability 1-*p* used to misdirect packets. When *p* is 1, the algorithm degrades simply to the Shortest Path algorithm. As *p* decreases however, more packets are misdirected, and in cases when the network contains hot nodes, performance can be gained.

In the remainder of this section we will concentrate on the mean packet delay as the target measure of performance.

To select the best value of *p* for a particular case, we measured the mean packet delay, independent of the number of hops, for several probabilities p, and for several possible number of hot spots. The resulting graph is shown in Figure 8. The jagged shape of the curves may be due to the fact that the mean delay is computed as the average of all the

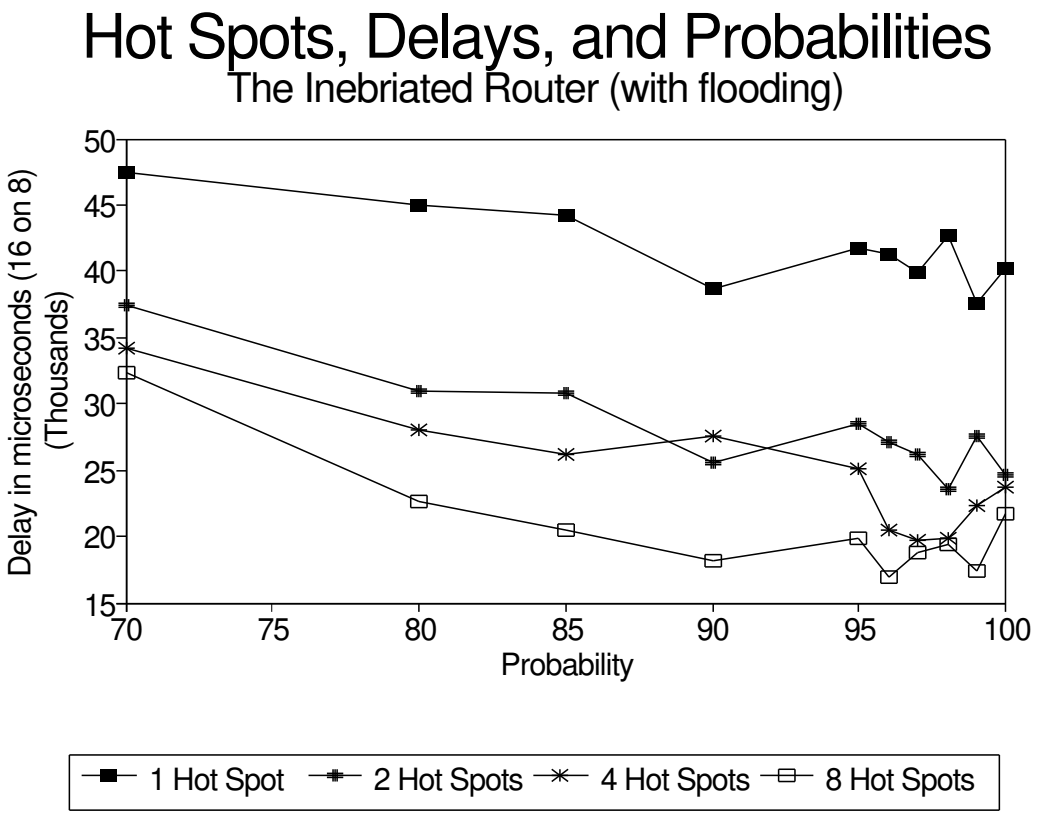

Figure 8: Mean packet delay as a function of number of hot spots and probability *p* of misdirection.

delays, independent of the number of hops. We will explore a more refined analysis taking into account the hops in future work.

It is interesting to note, as we would expect, that all four curves seem to exhibit a minimum. When *p* is 1, the path selected is always the shortest one, and the delays measured are shortest-path delays. When *p* decreases, however, messages start to suffer misdirection, and the farther they go, the more likely they will experience misdirection. As a result, the paths start to lengthen, at random, and the congestion at nodes close to the hot spots is reduced. The mean delays decrease. But, as *p* becomes even smaller, the packets get misdirected too often, to the point that some may travel a long time before they finally "hit" their destination. The delays start increasing.

Therefore, we can expect that for each hot-spot configuration there exists a best probability that yields the lowest delays, and Figure 8 is an indication that such a probability exists.

To test the performance of the inebriated router algorithm against the other two randomized routing algorithms of Valiant and Greenberg, we selected the best probability for each hot spot configuration and run our experiments, measuring the mean packet delays for all three algorithms.

Figure 9 shows the mean packet delays for the three randomized algorithms when the network is flooded with packets going to 1, 2, 4 or 8 hot spots in a 4-dimensional hypercube.

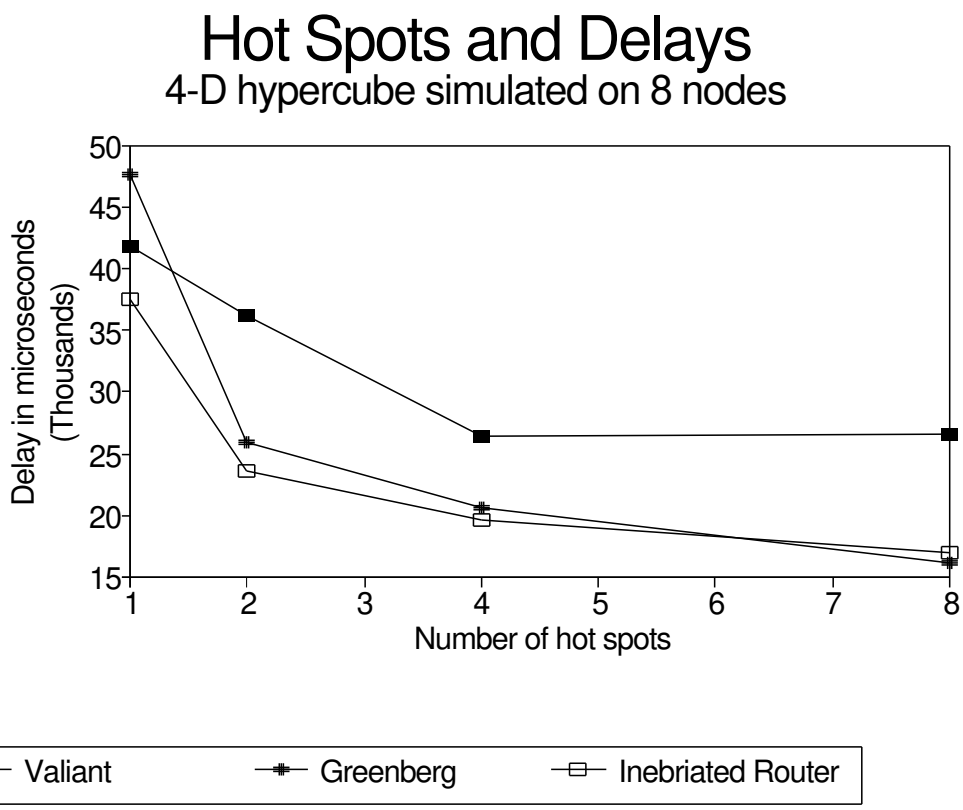

Figure 9: Mean delays for the three randomized algorithms in the presence of 1, 2, 4, and 8 hot spots in a 4-dimensional hypercube.

Two interesting observations can be made. The first one is that both the Inebriated Router and Greenberg perform better than Valiant for 2 or more hot spots. When the network sends messages to 1 hot-spot, the Greenberg algorithm performs worst. The possible explanation is that the queues leading to the hot spot start becoming non empty early on, which forces Greenberg and Hajek's routers to create an artificial flow of packets in the colder part of the network[3]. This results in packet exhibiting longer average delays.

The second observation is that the Inebriated Router algorithm performs better than Greenberg for 4 and fewer hot spots, and slightly worse for 8 hot spots. This is good news for the proposed algorithm, but the reader should remember that for each network hotness, we tuned our algorithm by selecting the best probability for our algorithm, while Greenberg dynamically adjusts its routing to avoid contention. Nevertheless, the figure shows that the mean packet delay can be reduced with our scheme.

4.5 Deflection of packets from the shortest path

The last measure of interest is the deflection of packets from their shortest path. Since the randomized routing schemes introduce randomness in the routing, some packets may find themselves circulating not only far from their ideal route, but, as we saw in the previous section in the case of the Greenberg algorithm, may be kept circulating in the colder part of the network until the congestion is removed around the hot nodes.

---

[3]Greenberg's original work considers only queues of size 1, and the router selects a longer path for a packet when the queues leading to the shortest paths are full. In our case, since we have queues with capacities larger than 1, we adopted non empty queues as the threshold for the router. It is possible that a different threshold may improve the observed performance of Greenberg and Hajek's algorithm in this case.

We compute the deflection as the difference between the actual number of hops taken by a packet and the number of hops on the shortest path. Figure 10 shows the histogram of the distribution of the amount of deflection incurred by packets for the four algorithms. The network is a 4-dimensional hypercube mapped onto a 3-D hypercube. By definition, the Shortest path algorithm exhibits no deflection, which translates in the histogram in a 100%

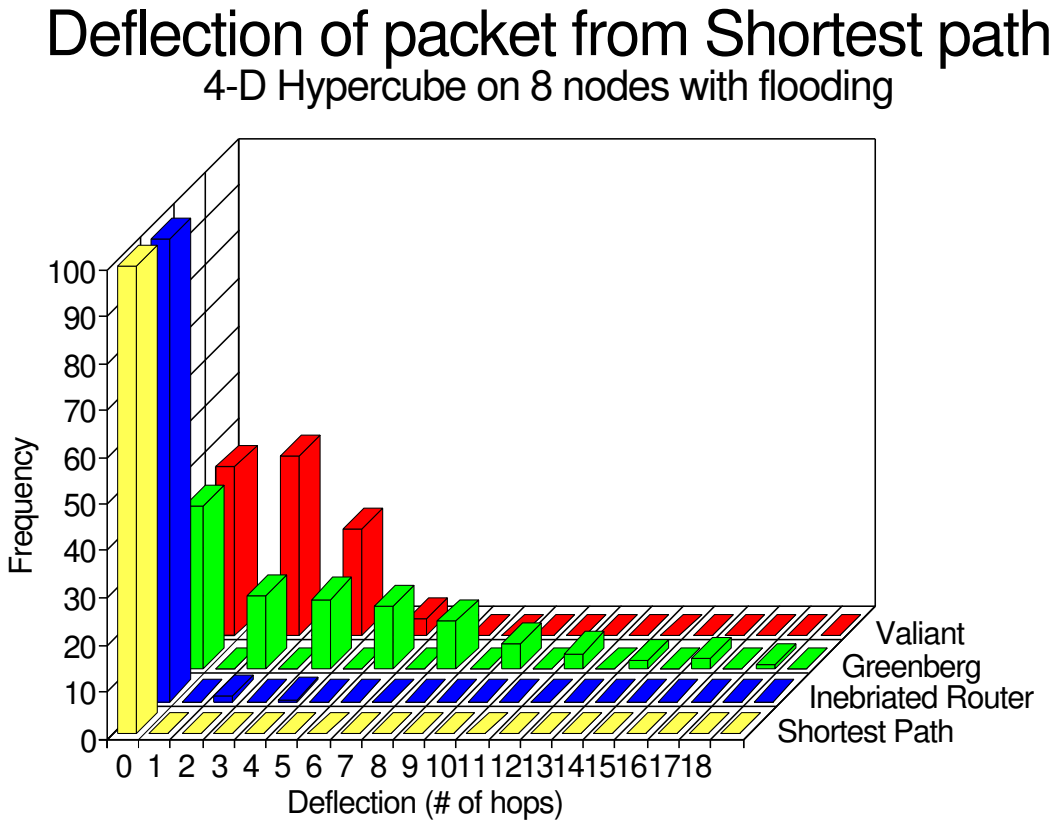

Figure 10: Deflection of packets from their shortest path.

bar for 0 hops. For the network considered and the number of hot spots (only 1), the Inebriated router exhibits almost no deflection from the shortest path. This is a particular case due to the high probability p that had been chosen for this network and degree of hotness.

The interesting data in this histogram, however, appear in the deflection for the Greenberg and Valiant algorithms. We see here the justification of our previous remark on Greenberg and Hajek's algorithm in networks with a small number of hot spots. A significant number of packets are deflected far from their path. In a 4-dimensional network, the shortest path may contain 1, 2, 3 or 4 hops. The data shown in Figure 10 shows that some packets may go through 18 additional hops on their journey to their destination under Greenberg and Hajek's algorithm. Not unexpectedly, Valiant also exhibits significant deflection due to the two-phase approach, but this deflection is limited, since a packet always uses the shortest path in the first (randomized) and second phase. Note that in both algorithms the deflection is always an even number, since each "wrong" turn taken by a packet on its way to its destination must be negated by another "right" turn to bring it back to a shortest path, due to the network topology.

5 Conclusions

We have presented a new randomized routing algorithm for message passing networks, and we have measured the performance of the algorithm on a transputer network. The algorithm was compared to two previously published schemes by Greenberg and Hajek, and by Valiant. The implementation of the experiments along with the data recorded suggest several interesting points.

1. When the physical hypercube hosts a larger logical hypercube, the performance statistics vary differently. The mean packet delay seems to vary linearly, with a slope depending on two factors: the degree of multitasking required of each physical node (transputer), and the reduced communication time when on-chip communication occurs. The mean queue size and node population, on the other hand, showed similar histograms when algorithms are compared.
2. The Inebriated Router algorithm outperforms the other randomized routing algorithms when the network exhibits a few very hot nodes. In this case, the Inebriated Router introduces enough randomness in selecting the path that the traffic of packets around the hot-spots is reduced, resulting in the shortest mean packet delays. It seems that the same conditions create a worst case routing for the Greenberg algorithm which keeps packets circulating away from the hot spots, hence draining nodes from additional computation time.

Several areas call for improvement and future work. One of them is a better selection of the probability p. The current implementation requires the user to measure the best probability for a particular network, and a specified number of hot spots. This method is currently rather crude and we are in the process of refining it. One of the attractive aspects of the Greenberg algorithm is it dynamically adapts to the flow of packets. Such a feature would be useful for the Inebriated Router algorithm.

Maybe the most likely area that we shall concentrate on is the extension of our study to larger hypercubes, of degree five and higher, and on a more thorough analysis of the influence of the folding on the different performance measures. Although this extension is theoretically a simple change of a few constants in our program (e.g., *LMAX*), we have found that the size of the code and data structures stored at each node quickly exhausts the one megabyte of memory currently associated with each transputer. Our next step will thus be to trim the code and fine tune the allocation of memory to processes to allow the cohabitation of several logical nodes on each transputer.

The code for our program, along with this report can be obtained by sending email to smitra@sophia.smith.edu or to thiebaut@sophia.smith.edu, or by anonymous ftp from grendel.smith.edu, directory pub/transputer.

Acknowledgment

We would like to acknowledge Chris Nevison, Nan Schaller and Paul Tymann for their NSF sponsored workshop held at Colgate University in the summer 1992, during which the idea of the inebriated router was triggered by an enlightening talk by Tom Leighton. Thanks also to Tom Leap and Dino Schweitzer who participated in generating the code for an early version of the algorithm presented here.

References

[1] Aiello, W. A., Leighton, F. T., Maggs, B. M., and Newman, M. "Fast algorithms for bit-serial routing on a hypercube," *Mathematical Systems Theory*, Vol. 24, No. 4, pp. 253-271, 1991.

[2] Baccelli, F., Gelenbe, E., and Plateau, B. "An end- to -end approach to the resequencing problem," *J. ACM*, Vol. 31, No. 3, pp. 474-485, July 1984.

[3] Greenberg, A., and B. Hajek, "Deflection routing in hypercube networks," *IEEE Trans. Comm.*, Vol. 40, No. 6, pp. 1070-1081, June 1992.

[4] Leighton, T., B. Maggs, and S. Rao, "Universal Packet Routing Algorithms," in 29th *Annual Symposium on Foundations of Computer Science*, IEEE, pp. 256-271, October 1988.

[5] May, D., and Thompson, P., "Transputers and Routers: Components for concurrent machines," *Proceedings of the 3rd Transputer/Occam International Conference*, Tokyo IOS Press, May 1990.

[6] Mitra, D., and Cieslak, R. A. "Randomized parallel communications on an extension of the Omega-network," *J. ACM*, Vol. 34, No. 4, pp. 802-824, October 1987.

[7] Peleg, D., and E. Upfal. "A time-randomness trade- off for oblivious routing," *Siam J. Comput.*, Vol. 19, No. 2, pp. 256-266, April 1990.

[8] Valiant, L. G., "A scheme for fast parallel communication," *SIAM J. Comput.*, Vol. 11, No. 2, pp. 350-361, May 1982.