Routing in Networks

A. Borodin *
J.E. Hopcroft*

October 1981

TR 81-477

Department of Computer Science
Cornell University
Ithaca, New York 14853

---

* University of Toronto
** Cornell University
This research was supported in part by ONR contract N00014-76-C-0018.

Routing in Networks

A. Borodin[1]

J.E. Hopcroft[2]

Introduction

The problem of routing packets in a network arises in a number of situations. Various routing protocols have been considered. For packet switching networks we are concerned with arbitrary interconnections. For these general networks, protocols such as forward state controllers have been studied. For applications in parallel computer architecture we are concerned with special networks such as a d-dimensional hypercube or a shuffle exchange interconnection. In this setting $O(\log n)$ global strategies are know as are $O(\log^2 n)$ local strategies.

For our purposes, a network is a diagraph $\langle V,E \rangle$ where the set of nodes V are thought of as processors, and $(i,j) \in E$ denotes the ability of

---

[1] University of Toronto

[2] Cornell University

This research was supported in part by ONR contract N00014-76-C-0018.

processor i to send one packet or message to processor j in a given time step. A packet is simply an <origin,destination> pair or, more generally, <origin,destination,bookkeeping information>.

A set of packets are initially placed on their origins, and they must be routed in parallel to their destinations; bookkeeping information can be provided by any processor along the route traversed by the packet. The prototype question in this setting is that of full permutation routing, in which we must design a strategy for delivering N packets in an N node network, when π: {origins} $\rightarrow$ {destinations} is a permutation of the node labels {1,2,...,N}. Other routing problems, in particular partial routing where π is L_1 but there may be less than N packets, are also of immediate interest. Unless stated otherwise, a routing strategy will denote a solution to the full permutation routing problem.

There are three positive routing results which provide the motivation for this paper.

1. Batcher's (see Knuth [2]) $O(\log^2 N)$ sorting network algorithm (say, based on the bitonic merge) can be implemented as a routing strategy on various sparse networks, such as the shuffle exchange (Stone [6]), the n-dimensional cube (Valiant and Brebner [7]), or the cube connected cycles (Preparata and Vuillemin [4]). This constitutes a local or distributed strategy, in that each processor p decides locally on its next action using only the packets at p. Indeed, the algorithm can be implemented so that there is exactly one packet per processor throughout execution. We also note that Batcher's bound is a worst case bound, holding for any initial permutation of the packets; in fact, every initial permutation uses

the same number of steps.

2. Valiant and Brebner [7] construct an O(log N) Monte Carlo local routing strategy for the n-cube. In a Monte Carlo strategy, processors can make random choices in deciding where to send a given packet Valiant's strategy achieves O(log N) in the sense that for every permutation, with high probability (eg $\geq 1-N^c$) all packets will reach that destination in $\leq (1+c)$ log N steps. Valiant's analysis can also be used to show that for a random input placement, the "naive strategy" on the n-cube terminates in O(log N) steps with high probability (here, the probability is on the space of possible inputs). The naive strategy is to order the dimensions of the cube, and then proceed to move packets along those dimensions (in order) in which the origin and destination differ.

3. The Slepian-Benes-Clos permutation network (see Lev, Pippenger and Valiant [3]) can be implemented as a global worst case O(log N) routing scheme (on any of the above mentioned sparse networks). Here, a pair of processors at a given time step simulate a switch of the permutation network; as such, the actions of any processor depend on the entire permutation.

We note that each of these strategies can be modified to handle partial routing. (This is immediate except for the strategy derived from Batcher's algorithm.) The obvious question remains as to whether or not there exists a local, worst case, O(log N) routing strategy. In order to make the distinction between local and global more meaningful we must limit the amount of bookkeeping information (say to O(log N) bits) since otherwise we can funnel all the input permutation

information to a distinguished processor which then broadcasts this information throughout the network.

The Relation Between Fixed Connection and Global Memory Models

Before proceeding to discuss routing algorithms for fixed connection networks, we want to briefly relate such parallel machines with parallel models based on a global memory. Indeed, this relation is yet another motivation for the importance of the routing problem. The importance of the routing problem is also emphasized in the paper of Galil and Paul [1] who consider simulations between various parallel models. We mention only a few global memory models:

1. PRAC (Lev, Pippenger and Valiant) - Simultaneous read or write (of the same cell) is not allowed.

2. PRAM (Fortune and Wyllie) - Simultaneous fetches are allowed but no simultaneous writes.

3. WRAM - WRAM denotes a variety of models that allow simultaneous reads and (certain) writes, but differ in how such write conflicts are to be resolved.

   a. (Shiloach and Vishkin) a simultaneous write is allowed only if all processors are trying to write the same thing, otherwise the computation is not legal.

   b. An arbitrary processor is allowed to write

   c. (Goldschlager) the lowest numbered processor is allowed to write.

For the purpose of comparison with an n-node fixed connection network, we assume the global memory models have n processors. It is obvious that even the weakest of the above models, the PRAC, can efficiently simulate a fixed connection network by dedicating a memory location for each directed edge of the network. Conversely, Lev, Pippenger and Valiant [3] observe that a fixed connection network capable of (partial) routing in r(n) time, can simulate one step of a PRAC in time O(r(n)). With some extra care, one can also simulate one step of a PRAM in time O(r(n)). The idea is roughly as follows:

a) Sort the read requests into consecutive locations; that is, sort pairs <request for memory located in i, by processor j> by i and then j. We note that other authors (see Schwartz [5] for the Ultracomputer) have indicated that sorting can often be used to implement partial routing which is all that is required here. The omitted details are not difficult.

b) Fan in all requests for a location i to a specified processor. This is done in such a way that a given processor only has to remember to which neighbors the requested information must be returned. This step will also be dependent on the network and will take either $O(\log n)$ or $O(\log^2)n$ time.

c) In the specified processors do a memory fetch (i.e. this is another partial sort).

d) The requested information (the memory contents) is distributed back to requesting processors by the specified processor.

Obviously, this also gives a way to simulate a PRAM by a PRAC; we do not know of a more efficient direct simulation. Finally, we note that a p-processor PRAM can simulate a step of a p-processor WRAM by running a tournament at a cost of O(log p). By these observations, and using Batcher's sort, one sees that all these parallel models have time complexities within a multiplicative $\log^{\alpha} n$ factor.

A lower bound for a special case

It turns out to be surprisingly difficult to analyze reasonable simple strategies. We are able to show, however, that a very simple class of strategies, including the naive strategy, cannot work well in the worst case, this being the case for a wide class of networks. Specifically, we study oblivious strategies where the route of any packet is completely determined by the <origin,destination> of the packet. Oblivious strategies are almost, but not quite, local by definition; we might still determine when a processor sends a packet along an edge by global means.

In this section we show that for any oblivious routing protocol for a network of n processors in which the maximum number of processors directly connected to any processor is d, there exists a permutation that requires time $\sqrt{n}/d^{\frac{3}{2}}$. In particular, for an n-cube there exists a permutation that requires $\sqrt{n}/(\log n)^{\frac{3}{2}}$ For ease in understanding we first prove the lower bound for a more restricted class of protocols, namely those where the next step in the route of a packet depends only on its present location and its final destination. For this class the

set of routes for a packet heading for a given destination forms a tree. To see this observe that at any vertex there is a unique edge that the packet will take. Following the sequence of edges from any vertex must lead to the final destination.

If we superimpose the n trees determined by the n possible destinations for packets, each vertex is on n trees. This suggests that we might be able to route n packets through a vertex. Since at most d packets can leave the vertex at any given time this would imply a delay of n/d. The difficulty is that in order to force a packet headed for a given destination to go through vertex v we must initially start the packet on a vertex that is a descendent of v in the particular destination tree. But there may be only a small number of such descendents of v and if many trees have the same set there will not be enough descendents to start each packet at a distinct vertex. For this reason we wish to consider only vertices in destination trees that have a large number of descendents. This motivates the following technical lemma.

Lemma: Let $T_{d,k}(n)$ be the minimum number of vertices with k, k>=2, or more descendents in any n vertex tree with maximum degree d, d>=2. Then

$$T_{d,k}(n) \geq \frac{n-k+1}{1+(d-1)(k-1)}$$

Proof: The proof is by induction on n. The lemma is clearly true for n<k. Let T be a tree with n>=k vertices. T consists of a root plus s subtrees where s<=d. Let the ith subtree have n_i vertices. Then, by the induction hypothesis, T has at least

$$1+\sum_{i=1}^{s} T_{d,k}(n_i) = 1+\sum_{i=1}^{s} \frac{n_i-k+1}{1+(d-1)(k-2)}$$

$$\leq 1+\sum_{i=1}^{s} \frac{n_i-k+1}{1+(d-1)(k-1)}$$

vertices with at least k descendents. Since $\sum_{i=1}^{s} n_i = n-1$ this sum is

$$1+\frac{n-s(k-1)-1}{1+(d-1)(k-1)} = \frac{n-k+1}{1+(d-1)(k-1)} + 1 - \frac{1+(s-1)(k-1)}{1+(d-1)(k-1)}$$

$$\geq \frac{n-k+1}{1+(d-1)(k-1)}$$

In each destination tree mark those vertices that have at least k descendents. Let $k = \sqrt{\frac{n}{d}}$. In the network assign a count to each vertex indicating the number of destination trees in which the vertex is marked. Since at least $\frac{n}{dk}$ vertices are marked in each tree, the sum of the counts must be at least $\frac{n^2}{dk}$. Therefore, the average count (over all vertices) is at least $\frac{n}{dk} = \sqrt{\frac{n}{d}}$ which implies there is at least one vertex v_0 which has at least $\sqrt{\frac{n}{d}}$ descendents in each of $\sqrt{\frac{n}{d}}$ descendent trees. For each of these descendent trees we can place the corresponding packet at some vertex of the network so that it will pass through vertex v_0. Thus $\sqrt{\frac{n}{d}}$ packets will go through v_0. Since v_0 is of degree at most d, it requires time at least equal to $\sqrt{\frac{n}{d^3}}$. In particular any routing procedure for an n-cube where the route of a packet depends only on the destination requires time at least $\sqrt{n}/(\log n)^{\frac{3}{2}}$.

A lower bound for oblivious routing

The above proof can be modified to apply to oblivious routing schemes. Consider a single destination. For each source construct the path for a packet headed for the destination. We no longer have a tree since the route of a packet depends on the source as well as the destination. However, we can prove a lemma analogous to the above lemma for trees. We modify the lemma to say that there must be at least $T_{d,k}(n)$ vertices having at least k paths through them. The inductive hypothesis is that in any directed graph with maximum fan-in d, with n loop-free directed paths to a designated vertex, at least $T_{d,k}(n)$ vertices must be on at least k paths. The lower bound for oblivious routing then follows exactly as in the special case.

An oblivious routing algorithm for an n-cube

The question remaining is how tight is the lower bound. The answer depends on the actual structure of the network. One important parameter in addition to the degree is the diameter of the graph. Clearly if the diameter of the graph is n we cannot hope for a $O(\sqrt{n})$ algorithm. However, even for some O(log n) diameter graphs with degree 2 we cannot achieve an $O(\sqrt{n})$ algorithm since there may be an isthmus or other type of bottleneck. However, for many structures there are oblivious routing algorithms that are close to this lower bound.

Consider a d-dimensional hypercube. The following oblivious algorithm will route packets for any permutation in time $O(\sqrt{n})$. Since the hypercube has maximum degree log n this is close to the lower bound of $\sqrt{n}/(\log n)^{\frac{3}{2}}$. Order the dimensions $x_1, x_2, \ldots, x_{\log n}$. At time 1

transmit any packet whose destination is on the opposite subcube determined by the x_1 dimension. This may result in as many as two packets being on a vertex. Next transmit packets across the x_2 dimension, etc. After $\frac{1}{2}\log n$ dimensions there may be as many as $\sqrt{n}$ packets at a vertex causing a delay of $\sqrt{n}$. In each subsequent step the maximum number of packets that can be on a vertex halves, with eventually each packet arriving at its destination. It seems possible to improve this bound to $\sqrt{n}/\log n$ if one partitions the $\sqrt{n}$ vertices that could go through a vertex v_0 into log n groups and routes them by different edges.

Future work

Another restriction on routing is minimality. A minimal routing scheme forbids transmitting along an edge if it increases the distance of the packet from its destination. Thus every packet must follow a shortest path. For minimal routing schemes it is an interesting open problem whether there is a local (or even a global) routing scheme that is $O(\log^r n)$ for any r. For regular networks such as the n-cube we know of no monotone scheme better than $O(\sqrt{\frac{n}{\log n}})$.

Another interesting question is whether there is a local routing algorithm for say an n-cube that is better than $O(\log^2 n)$. In particular, does the following algorithm or some variant of it, route an arbitrary permutation in $O(\log n)$. Consider some vertex. At a given stage as many as d packets, where d is the dimension of the cube, will arrive. As many as possible will be sent closer to their destinations. The remaining packets will be shipped to vertices of distance one greater. Since packets are not allowed to build up at a vertex the effect is to enlarge bottlenecks to several vertices and hence to allow more packets

to get to their destinations in a given time. Although the algorithm appears promising we have no analysis better than $O(n)$, $n=2^d$.

We note that the above strategy avoids queues. It is also an interesting question to study the class of strategies which do not use queues (like Batcher).

References

[1] Galil, Z., and W.J. Paul. An efficient general purpose parallel computer. Proceedings 13th Annual ACM Symposium on Theory of Computing, Milwaukee, Wisconsin, (1981), 247-256.

[2] Knuth, D.E. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts, 1972.

[3] Lev, G., N. Pippenger and L.G. Valiant. A fast parallel algorithm for routing in permutation networks. IEEE Transactions on Computers (1981).

[4] Preparata, F.P., and J. Vuillemin. The cube-connected cycles. Proceedings 20th Symposium on Foundations of Computer Science (1979), 140-147.

[5] Schwartz, J.T. Ultracomputers. ACM TOPLAS 2 (1980), 484-521.

[6] Stone, H. Parallel processing with the perfect shuffle. IEEE Transactions on Computers C20:2 (1971), 153-161.

[7] Valiant, L.G., and G.J. Brebner. Universal schemes for parallel computation. Proceedings 13th Annual ACM Symposium on Theory of Computation, Milwaukee, Wisconsin (1981), 263-277.