May 1989

General Routing Algorithms for Star Graphs

Michael A. Palis
University of Pennsylvania

Sanguthevar Rajasekaran
University of Pennsylvania

David S.L. Wei
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-35.

General Routing Algorithms for Star Graphs

Abstract

In designing algorithms for a specific parallel architecture, a programmer has to cope with topological and cardinality variations. Both these problems always increase the programmer's effort. However, an ideal shared memory abstract parallel model called the parallel random access machine (PRAM) [KRUS86, KRUS88] that avoids these problems and also simple-to-program has been proposed. Unfortunately, the PRAM does not seem to be realizable in the present or even foreseeable technologies. On the other hand, a packet routing technique can be employed to simulate the PRAM on a feasible parallel architecture without significant loss of efficiency. The problem of routing is also important due to its intrinsic significance in distributed processing and its important role in the simulations among parallel models.

The routing problem is defined as follows: Given a specific network and a set of packets of information in which a packet is an (*origin, destination*) pair. To start with, the packets are placed on their origins, one per node. These packets must be routed in parallel to their own destinations such that at most one packet passes through any link of the network at any time and all packets arrive at their destinations as quickly as possible. We are interested in a special case of the general routing problem called *permutation routing* in which the destinations form some permutation of the origins. A routing algorithm is said to be *oblivious* if the path taken by each packet is only dependent on its source and destination. An oblivious routing strategy is preferable since it will lead to a simple control structure for the individual processing elements. Also oblivious routing algorithms can be used in a distributed environment. In this paper we are concerned with only oblivious routing strategies.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-35.

GENERAL ROUTING ALGORITHMS FOR STAR GRAPHS

Michael A. Palis,
Sanguthevar Rajasekaran
and David S. L. Wei

MS-CIS-89-35
LINC LAB 153

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

May 1989

Acknowledgements: This research was supported in part by DARPA grant NOOO14-85-K-0018, NSF grants MCS-82-07294, DCR-84-10413, MCS-83-05221, MCS-8219196-CER, IRI84-10413-AO2 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

General Routing Algorithms for Star Graphs

1 Introduction

In designing algorithms for a specific parallel architecture, a programmer has to cope with topological and cardinality variations. Both these problems always increase the programmer's effort. However, an ideal shared memory abstract parallel model called the parallel random access machine (PRAM) [KRUS86, KRUS88] that avoids these problems and also simple-to-program has been proposed. Unfortunately, the PRAM does not seem to be realizable in the present or even foreseeable technologies. On the other hand, a packet routing technique can be employed to simulate the PRAM on a feasible parallel architecture without significant loss of efficiency. The problem of routing is also important due to its intrinsic significance in distributed processing and its important role in the simulations among parallel models.

The routing problem is defined as follows: Given a specific network and a set of packets of information in which a packet is an $\langle origin, destination \rangle$ pair. To start with, the packets are placed on their origins, one per node. These packets must be routed in parallel to their own destinations such that at most one packet passes through any link of the network at any time and all packets arrive at their destinations as quickly as possible. We are interested in a special case of the general routing problem called *permutation routing* in which the destinations form some permutation of the origins. A routing algorithm is said to be *oblivious* if the path taken by each packet is only dependent on its source and destination. An oblivious routing strategy is preferable since it will lead to a simple control structure for the individual processing elements. Also oblivious routing algorithms can be used in a distributed environment. In this paper we are concerned with only oblivious routing strategies.

Both deterministic and randomized schemes have been studied in solving routing problems ([VALI81] [VALI82] [UPFA84] [PIPP84] [RANA87] [LEIG88a, LEIG88b] [ALT87] [BORO82] [KRIZ88]). However, most of the work has focused on bounded degree networks, such as cube-connected cycles (CCC), butterfly, shuffle-exchange, and mesh, etc. Some research work has also been done on a binary n-cube (hypercube) which is not a bounded degree network. All of these networks (except the mesh) have **logarithmic** diameter and have randomized routing algorithms that run in **logarithmic** time. Clearly, these algorithms are optimal. An interesting open question is that can we do optimal routing on a network with **sublogarithmic** diameter?

In this paper we settle this question in the affirmative. In particular, we present an optimal randomized oblivious routing algorithm for the *star graph* ([AKER87, AKER86]) which has **sublogarithmic** diameter.

The picture is quite different for the case of oblivious deterministic routing strategies. Borodin and Hopcroft [BORO82] have shown that for any graph of N nodes with degree d, the maximum delay, in the worst case, of any oblivious deterministic routing scheme is $\Omega(\sqrt{\frac{N}{d}})$(This lower bound assumes that each node can process only one packet at a time, regardless of the number of incoming and outgoing links). We present an oblivious deterministic routing algorithm for the star graph. This algorithm runs in $O(\sqrt{N})$ time with $O(\sqrt{N})$ queue size, where N is number of nodes in the graph. We conjecture that the lower bound for the star graph is $\Omega(\sqrt{N})$ and that our algorithm is optimal.

2 An oblivious deterministic routing algorithm for the n-star graph

2.1 The star graph

Definition 1 Let $d_1 d_2 \ldots d_n$ be a permutation of n symbols, *e.g.*, $1 \ldots n$. For $1 < j \leq n$, we define $SWAP_j(d_1 d_2 \ldots d_n) = d_j d_2 \ldots d_{j-1} d_1 d_{j+1} \ldots d_n$.

Definition 2 An n-star graph is a graph G=(V,E) with $\mid V \mid = n!$ nodes, where V = $\{d_1 d_2 \ldots d_n \mid d_1 d_2 \ldots d_n$ is a permutation of $1...n\}$, and E = $\{(u,v) \mid u,v \in E$ and $v = SWAP_j(u)$ for some $j, 1 < j \leq n\}$.

The 4-star graph is depicted in Figure 1.

In [AKER87], Akers, Harel and Krishnamurthy have shown that the star graph is superior to the n-cube with respect to the degree and diameter. An n-star graph has $n!$ nodes, degree $n-1$, and diameter $\lfloor \frac{3}{2}(n-1) \rfloor$. On the other hand, an n-cube has 2^n nodes, degree n, and diameter n. Thus, the degree and diameter of the star graph grows more slowly as a function of the network size than does the n-cube. Moreover, the star graph is both vertex symmetric and edge symmetric (just like the n-cube). Oftentimes, these properties lead to a simpler analysis of the routing algorithm.

In [AKER86, AKER87], an algorithm was presented for routing a *single* packet from a source to an arbitrary destination. The more general problem of permutation routing was not considered. In the next two sections, we present efficient deterministic and randomized algorithms for permutation routing on the star graph. Both algorithms are oblivious.

Definition 3 A subgraph of an n-star graph G is said to be an *i-th stage subgraph*, denoted G^i, *iff* G^i is itself an $(n-i)$-star graph, $0 \leq i < n$, and the last i symbols of the labels of all nodes in it are identical.

The G^i's of any G^{i-1} partition it into $n-i+1$ identical subgraphs. Let's define the *stage* of the network during a run of the routing algorithm to be simply the collection of the nodes together with the packets each node holds in its queue. Hence the routing algorithm can be thought of as a sequence of stage transitions $S_1, ..., S_f$, where in S_1 each node has a single packet that originated in that node, and in S_f each node has a single packet that is destined for it.

Look at all the G^i's that constitute any G^{i-1}. It is easy to see that for any node u in any one of these G^i's, there is exactly one other node v adjacent to u such that v is contained in some other G^i. We call v the **critical point** to u and vice-versa, at stage i. For example, in Figure 1(b), node $BACD$ is a critical point to node $DACB$ at stage 1.

Definition 4 A stage S_i is said to be *i-th stage stable*, denoted S^i_{stable}, *iff* for every *i-th stage subgraph* G^i, the destination of each packet in the subgraph is in the subgraph itself.

2.2 An oblivious deterministic routing algorithm

The routing scheme is based on divide-and-conquer. The algorithm runs in stages. In the first stage each packet is sent to the G^1 (refer to Definition 1) it belongs to. In the second stage each packet is sent to the G^2 it belongs to, and so on. Finally, in stage $n-1$, each packet is sent to the G^{n-1} it belongs to (which is the single node destination of the packet). Thus our routing scheme can be viewed

as a sequence of stage transitions $S^0_{stable}S^1_{stable}\cdots S^i_{stable}S^{i+1}_{stable}\cdots S^{n-1}_{stable}$. Also our algorithm is such that once the algorithm enters S^i_{stable} (for any i), it will also be in S^j_{stable} for $j \leq i$. Once the routing reaches S^{n-1}_{stable}, the task will be complete. The formal description of the routing scheme is shown in Algorithm A.

Algorithm A
{Each node has two queues Q_1 and Q_2.
Initially, Q_1 has a single packet that originates from the node, and Q_2 is empty.
The second **for** loop stands for transition from S^{i-1}_{stable} to S^i_{stable} (for $i = 1, ..., n-1$).}
for every node $\pi = d_1 d_2 \ldots d_n$ *in parallel* **do**
  for each $1 \leq i < n$ **do**
    Append Q_2 to the tail of Q_1;
    for $j := 1$ to $\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)$ **do**
      Let x be the packet at the head of Q_1 and let $\overline{d}_1\overline{d}_2 ... \overline{d}_n$ be the address of this packet's destination.
      {The algorithm is now in S^{i-1}_{stable}.
      From Definition 3, we know that $d_{n-i+2}d_{n-i+3}...d_n$ is identical to $\overline{d}_{n-i+2}\overline{d}_{n-i+3}...\overline{d}_n$}.
      if $d_{n-i+1} = \overline{d}_{n-i+1}$
      then Put x at the tail of Q_2;
        {So it could be processed in the next stage.
        Notice that x is already in the correct G^i}
      else
        if $d_1 = \overline{d}_{n-i+1}$
        then
          Send x to node $SWAP_{n-i+1}(\pi)$ to be appended to queue Q_1;
          {x will be in its correct G^i when it goes there }
        else
          Choose the unique j such that $d_j = \overline{d}_{n-i+1}$;
          Send x to node $SWAP_j(\pi)$ to be appended to Q_1;
          { When x reaches this node, it has to traverse one more link
          before it is in its correct G^i.}
end Algorithm A.

2.3 Performance analysis of Algorithm A

We will show that (1) $O\left(\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)\right)$ time is sufficient to make the transition from S^{i-1}_{stable} to S^i_{stable}, (2) the queue size for the algorithm is $O(\sqrt{n!})$, and (3) the run time of the whole algorithm is $O(\sqrt{n!})$. Let $\mathcal{M}_{q_i}$ be the maximum number of packets queued in any node during the transition from S^{i-1}_{stable} to S^i_{stable}. Clearly, the time needed for the transition from S^{i-1}_{stable} to S^i_{stable} is $O(\mathcal{M}_{q_i})$ since each packet needs only a constant amount of time to process.

Lemma 1 $\mathcal{M}_{q_i} \leq \min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)$

proof: (By induction)

Base Case: When $i = 1$, we have $\mathcal{M}_{q_i} \leq n$. This follows from the following fact. Suppose that we have two G^1's, say α and β, such that α and β are connected through several pairs of critical nodes*. Let one of them be (a_1, b_1) (See Figure 2). In the transition from S^0_{stable} to S^1_{stable}, the worst case of queuing for b_1 occurs when each node adjacent to a_1 wants to send its packet through a_1 to β and also a_1 wants to send its own packet through b_1 to β. Hence, including the packet that originally resided in b_1, we have a total of $(n-2)+1+1 = n$ packets that will pass through b_1. The same holds for the critical points of the other G^1's. But they are independent events, i.e. they will never affect each other.

Induction step: Suppose that Lemma 1 is true for $i = k$. We will prove it for $i = k+1$, i.e. we'll prove that $\mathcal{M}_{q_{k+1}} \leq \min\left(\prod_{s=1}^{n-k-1} s, \prod_{s=n-k}^{n} s\right)$.

Case A: $\mathcal{M}_{q_k} \leq \min\left(\prod_{s=1}^{n-k} s, \prod_{s=n-k+1}^{n} s\right) = \prod_{s=n-k+1}^{n} s.$

Fix any node b, and let a be the critical point to b at stage $k+1$. The only packets that will ever contribute to the queue size of b during the transition from S^k_{stable} to S^{k+1}_{stable} are those that ever reached node a or nodes adjacent to a which are in G^k. Since G^k is an $(n-k)$-star graph, a has $n-k-1$ other nodes adjacent to it (including b) in G^k. It follows, using the induction hypothesis, that the total number of packets that will reach b during the transition from S^k_{stable} to S^{k+1}_{stable} is at most $((n-k-1)+1)^*\left(\prod_{s=n-k+1}^{n} s\right)$ which is equal to $\prod_{s=n-k}^{n} s$. Notice that b is in a G^{k+1}. The queue size of b can not be greater than $\prod_{s=1}^{n-k-1} s$ because, only these many packets are destined for the G^{k+1} that b is in. (Figure 3 might help the reader better understand the proof.) Thus, we have $\mathcal{M}_{q_{k+1}} \leq \min\left(\prod_{s=1}^{n-k-1} s, \prod_{s=n-k}^{n} s\right)$.

Case B: $\mathcal{M}_{q_k} \leq \min\left(\prod_{s=1}^{n-k} s, \prod_{s=n-k+1}^{n} s\right) = \prod_{s=1}^{n-k} s.$

Clearly $\mathcal{M}_{q_{k+1}}$ is $\leq \prod_{s=1}^{n-k-1} s$, since there are only these many nodes in any G^{k+1} and hence only these many packets are destined for any G^{k+1}.

$$\text{Also } \mathcal{M}_{q_{k+1}} \leq \prod_{s=1}^{n-k-1} s \leq \prod_{s=1}^{n-k} s \leq \prod_{s=n-k+1}^{n} s \leq \prod_{s=n-k}^{n} s.$$

$$\text{Thus } \mathcal{M}_{q_{k+1}} \leq \min\left(\prod_{s=1}^{n-k-1} s, \prod_{s=n-k}^{n} s\right).\square$$

Theorem 1 The maximum queue needed in Algorithm A is

$$\max_i \left\{\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)\right\} = O(\sqrt{n!}).$$

*(a, b) is a pair of critical nodes if a is critical point to b and vice-versa.

Proof: Follows from Lemma 1 and the following fact. Given any integer N. Let $Z = \{(X,Y) : X$ and Y are integers and $X * Y = N\}$. Then $\max_{(X,Y)\in Z}\{\min(X,Y)\} \leq O(\sqrt{N})$.□

Theorem 2 A permutation routing in an n-star graph can be performed by an oblivious deterministic routing scheme in $O(\sqrt{n!})$ time steps.

Proof: Let $T(n)$ be the time steps needed for Algorithm A. From Lemma 1, we have

$$T(n) = \sum_{i=1}^{n-1}\left[\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)\right] < 4\sqrt{n!} = O(\sqrt{n!}).\square$$

Algorithm A always runs in time $O(\sqrt{n!})$. Even if a packet is very close to its destination at the beginning, it still has to go through $n-1$ stage transitions. This algorithm can be modified so that a packet can start its stage $k+1$ without waiting for other packets to finish their stage k. In such a modified algorithm each packet will carry along with it a $\log n$ bits information that corresponds to the stage the packet is in. This modification will result in a faster run time if every packet is close to its destination at the beginning. The modified algorithm follows. (In Algorithm A', *long time* means $O(\sqrt{n!})$ in the worst case but is considerably less for special cases.)

Algorithm A'
{To begin with each node has a single packet that originates in the node, and there is only one queue. Now each packet is in its G^0.}
for every node $\pi = d_1 d_2 ... d_n$ do the following **in parallel** for a *long time*
Let x be the packet at the head of π's queue and let $\overline{d}_1\overline{d}_2...\overline{d}_n$ be the address of this packet's destination.
Also let the packet x be in its G^i (realize that x carries i along with it).
{From the Definition 1, we know that $d_{n-i+1}d_{n-i+2}...d_n$ is identical to $\overline{d}_{n-i+1}\overline{d}_{n-i+2}...\overline{d}_n$}.
if $d_{n-i} = \overline{d}_{n-i}$
then Set $i = i+1$ and put x at the tail of π's queue;
{So it could be processed in the next stage.
Notice that x is already in the correct G^{i+1}}
else
if $d_1 = \overline{d}_{n-i}$
then
Set $i = i+1$ and send x to node $SWAP_{n-i}(\pi)$;
{x will be in its correct G^{i+1} when it goes there }
else
Choose the unique j such that $d_j = \overline{d}_{n-i}$;
Send x to node $SWAP_j(\pi)$;
{ When x reaches this node, it has to traverse one more link
before it is in its correct G^{i+1}.}
end Algorithm A'.

3 A randomized routing algorithm for the n-star graph

The large worst case delay of oblivious deterministic routing makes such schemes uninteresting from a practical point of view. But efficient routing algorithms that employ randomization have been discovered. For example, Valiant and Brebner [VALI81, VALI82] have given an $O(\log N)$ time oblivious

randomized routing scheme for the n-cube network, with $N = 2^n$ nodes. They use a two phase strategy in which packets are sent obliviously, first to random intermediate nodes and then to their correct destinations. They showed that there is a constant c'' such that every packet will reach its own destination in $\leq cc'' \log N$ steps with high probability (i.e. with probability $\geq 1 - N^{-c}$).

After Valiant's work, a lot of research on randomized routing ([ALEL82] [UPFA84] [RANA87] [LEIG88b] [KRIZ88]) has been done. But all these employ bounded degree networks such as butterfly, shuffle-exchange, d-way shuffle, and mesh, etc. The randomized routing lower bound for a bounded degree network is obviously $\Omega(\log N)$ because the diameter of a constant degree network is at least $\log N$. Thus, we won't be able to perform permutation routing on these networks in **sublogarithmic** time steps. The interesting question is: For those unbounded degree networks with **sublogarithmic** diameter, can we route (using randomization) a permutation request in **sublogarithmic** steps with high probability?

Valiant [VALI81] has shown that permutation routing can be done on the d-way shuffle graph (which has $N = d^n$ nodes and diameter n) in $O(n \log d / \log\log d)$ steps with high probability. For the n-way shuffle graph, Valiant's algorithm runs in time $O(n \log n / \log\log n)$ and hence is not optimal. In this section, we present a randomized routing algorithm for the n-star graph that runs in time of the order of the diameter with high probability.

We assume that all the links are bidirectional and also for each node there is a queue corresponding to each incoming and outgoing links. Furthermore, a node can receive a packet from each incoming link and send a packet along each outgoing link in one unit of time (this assumption has been made in [VALI81] also).

Algorithm B

Phase 1

Step 1: **for** each packet x **do in parallel** select a random intermediate node.

Step 2: *Use Algorithm* A' to send the packets to their intermediate random destinations.

{The queuing discipline is first-in first-out (FIFO).

A *long time* in Algorithm A', applied here, means $c''c\ n$ (for some c'' to be fixed in the analysis)}.

Phase 2

Use Algorithm A' to send each packet x from its intermediate node to its correct destination.

Analysis

Fact 1 The number of steps a packet x is delayed is less than or equal to the number of other packets that *overlap*[†] with x.

Proof: Refer to [VALI81].□

Fact 2 For any $n > 0$, there exists an i such that $\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right) = \prod_{s=1}^{n-i} s$ and $n - i > \frac{n}{2}$.

We can represent the stage transitions in our algorithm in the form of a *logical network*. A logical network is the following. Each column is simply the nodes in the network. The links from column $i-1$ to column i are the links (in the network) that can be used during the transition from S^{i-1}_{stable} to S^{i}_{stable}

[†]Two packets are said to overlap if there are ≥ 1 common links in their paths.

(in our algorithm). So a logical network represents the stage of the network at each time unit. Our proof will be simplified if it is given using the logical network. A logical network for the 3-star graph is shown in Figure 4. Since $n = 3$, we have only two stages (levels). Each node in column i has $n - i + 1$ incoming and $n - i$ outgoing links. Packets are delayed only in the case that more than one incoming links contain a packet and more than one of them must be forwarded to the same outgoing link.

Note that, as an example, if a packet x moving from node 123 to node 312 has to pass through node 213, it will never cause a delay to the packets in node 213 if the destination of those packets are not node 312. Also note that each link corresponds to at most 2 steps.

Theorem 3 For the n-star graph of $N = n!$ nodes, there exists a $c' > 0$ such that any permutation routing can be completed using Algorithm B in $4(c+1)n$ steps with probability at least $1 - \frac{8c'}{N^{c+1}}$[‡].

Proof: (A similar proof technique has been used by [RIVE87].)

Based on Fact 1, to determine the expected delay of a packet x, we only need to determine how many packets x' are expected to overlap with x. To simplify the discussion, let us first determine the probability that d packets overlap x's path for the first time in stage i. Consider a link, say L, in stage i. Based on Lemma 1, we know that these d packets can possibly originate from $\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)$ number of nodes. Thus, there are $\binom{\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)}{d}$ number of ways to choose the origins of these d packets. For each packet, there are $\prod_{s=n-(i+1)}^{n} s$ possible paths for the packet to take before it reaches stage $i+1$. Thus, the probability that all these d packets pass through link L is $\left(\frac{1}{\prod_{s=n-i-1}^{n} s}\right)^d$. Besides, the likelihood for the remaining $\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right) - d$ packets not to pass through link L is $\left(1 - \frac{1}{\prod_{s=n-i-1}^{n} s}\right)^{\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)-d}$. Hence, we have an upper bound for the probability that the number of packets, whose paths overlap a given path through link L for the first time at stage i, equals d. Let d_i be number of packets that delay a given packet for the first time in stage i. Then,

$$Prob(d_i = d) \leq \binom{\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)}{d} \left(\frac{1}{\prod_{s=n-i-1}^{n} s}\right)^d \left(1 - \frac{1}{\prod_{s=n-i-1}^{n} s}\right)^{\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)-d}$$

[‡]We will prove Theorem 3 only for Phase 1 and it will be clear how the proof can be modified to apply to the second phase as a mirror image of the first phase.

$$
\begin{aligned}
&\leq \begin{pmatrix} \min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right) \\ d \end{pmatrix} \left(\frac{1}{\prod_{s=n-i-1}^{n} s}\right)^d \\
&\leq \frac{\left[\min\left(\prod_{s=1}^{n-i} s, \prod_{s=n-i+1}^{n} s\right)\right]^d}{d!} \frac{1}{\left(\prod_{s=n-i-1}^{n} s\right)^d} \\
&\leq \frac{1}{\left(\left(\frac{n}{2}\right)^2\right)^d d!}, \text{by} \quad \text{Fact} \quad 2.
\end{aligned}
$$

But we are interested in the probability of a total delay d rather than the delay due to packets that meet the given packet for the first time in stage i. The total delay for the given packet is $\sum_i d_i$. This can be computed using generating functions.

The generating function for $Prob(d_i = d)$ is

$$G_i(x) = \sum_{d=0}^{\infty} \frac{\left(\left(\frac{n}{2}\right)^{-2}\right)^d}{d!} x^d = e^{\frac{x}{\frac{n^2}{4}}}$$

Since the generating function of a sum of random variables is the product of the individual generating functions, the generating function for $Prob(\sum_i d_i = d)$ is given by

$G(x) = \prod_{i=1}^{k} G_i(x) = e^{\frac{4k}{n^2}x} = \sum_{d=0}^{\infty} \left[\left(\frac{4k}{n^2}\right)^d \frac{1}{d!}\right] x^d$, where k is the number of stages in the algorithm.

Then the probability that the total delay is greater than a given amount, say δ, is:

$$
\begin{aligned}
Prob(\sum_i d_i \geq \delta) &\leq \sum_{d=\delta}^{\infty} \left(\frac{4k}{n^2}\right)^d \frac{1}{d!} \\
&\leq 2\left(\frac{4k}{n^2}\right)^{\delta} \frac{1}{\delta!} \\
&\leq 8\left(\frac{1}{n}\right)^{\delta} \frac{1}{\delta!}, \text{since} \quad k = n-1 \\
&\leq 8\frac{1}{(n^n)^c} \frac{1}{c!n!}, \text{let} \quad \delta = cn \\
&\leq 8c' \frac{1}{(n^n)^c} \frac{1}{n!}, \text{where} \quad c' = \frac{1}{c!} \\
&\leq 8c' \frac{1}{(n!)^{c+1}}. \square
\end{aligned}
$$

A similar proof technique can also be used to analyze the behavior of a simple but efficient randomized routing algorithm for the d-way shuffle. Our routing algorithm for the n-way shuffle achieves a better (in fact, optimal) time bound than that of [VALI81].

A d-way shuffle network has $N = d^n$ nodes. Each node can be labelled as $d_n d_{n-1} ... d_1$ where each d_i is a d-ary digit. A node labelled $d_n d_{n-1} ... d_1$ is connected to the nodes labelled $l d_n d_{n-1} ... d_2$ where l is an arbitrary d-ary digit. Therefore, the network has diameter n and a unique path of exactly n links

between any pair of nodes. If we choose $d = n$, then the network is an n-way shuffle. The following algorithm can be used to perform permutation routing on the n-way shuffle.

Algorithm C

Phase 1

Step 1: **for** each packet x **do in parallel** select a random intermediate node.

Step 2: Send the packets along the unique path to their intermediate random destinations.

{The queuing discipline is FIFO}

Phase 2

Send each packet x from its intermediate node to its correct destination along the unique path.

Theorem 4 For the n-way shuffle network of $N = n^n$ nodes, there exists a $c' > 0$, and an α, where $0 < \alpha < 1$, such that permutation routing can be performed using Algorithm C in $2(c+1)n$ steps with probability at least $1 - \frac{c'}{N^{c+\alpha}}$.

Proof: Similar to Theorem 3.□

4 Conclusions

An interesting open problem is if the lower bound of $\Omega(\sqrt{\frac{N}{d}})$ (of [BORO82]) can be improved for the star graph or the n-cube. We conjecture that $\Omega(\sqrt{N})$ is a lower bound for routing on both the star graph and the n-cube. If this were true then it will mean having **sublogarithmic** diameter doesn't help for oblivious deterministic routing strategies.

Valiant's two phase scheme has been proved to be a powerful technique for packet routing. Section 3 demonstrates that making use of generating functions to handle random variables can simplify the analysis of the behavior of the routing algorithm and can also lead to a tighter upper bound.

A deficiency with the state-of-the-art in packet routing is that the algorithms presented and their analysis are network-specific. The important open question is: Is there a *network-independent* routing algorithm that works for a large class of networks, rather than a specific network? A significant contribution in this direction has been reported very recently by Leighton, Maggs and Rao [LEIG88b]. They give a proof that any set of paths with distance d and congestion c can be off-line routed in $O(c + d)$ steps using constant-size queues. It is still open if the same time bound and queue size can be achieved for the case of on-line routing.

References

[AKER86] Akers, S. B. and B. Krishnamurthy, A group theoretic model for symmetric interconnection networks, *IIEEE Parallel Processing*, 1986, pp.216-223.

[AKER87] Akers, S. B., D. Harel, and B. Krishnamurthy, The star graph: an attractive to the n-cube, *IIEEE Parallel Processing*, 1987, pp.393-400.

[ALEL82] Aleliunas, R., Randomized parallel communication, PODC, Vol. 1, 1982, pp. 60-72.

[ALT87] Alt, H., T. Hagerup, K. Mehlhorn, and F. P. Preparata, Deterministic simulation of idealized parallel computers on more realistic ones, *ISIAM J. Comput.,* October, 1987, pp. 808-835.

[BORO82] Borodin, A. and J. E. Hopcroft, Routing, merging and sorting on parallel models of computation, in Proceedings of STOC, 1982, pp. 338-344.

[KRUS86] Kruskal, C. P., Algorithms for replace-add based paracomputers, *Proc. International Conference on Parallel Processing*, 1982, pp. 278-283.

[KRUS88] Kruskal, C. P., Rudolph, L. and Snir, M., A complexity theory of efficient parallel algorithms, *Proc. 15th International Colloquium on Automata, Languages, and Programming*, 1988, pp. 333-346.

[KRIZ88] Krizanc, D., Rajasekaran, S., and Tsantilas, Th., Optimal routing algorithms for mesh-connected processor arrays, *Proc. Third International Aegean Work Shop on Parallel Computation and VLSI Theory*, 1988. *Springer-Verlag Lecture Notes in Computer Science.*

[LEIG88a] Leighton, T., Makedon, F., and Tollis, I., A $2n-2$ step algorithm for routing in an n *times* n array with constant size queues, Unpublished manuscript, 1988.

[LEIG88b] Leighton, T., Maggs, B., and Rao, S., Universal packet routing algorithms, *IEEE Symposium on Foundations of Computer Science*, 1988, pp. 256-269.

[PAUL82] Beame, P., Random routing in constant degree networks, *Tech. Rept. no. 161/82*, Dept. of Compt. Sci., Univ. of Toronto.

[PIPP84] Pippenger, N., Parallel communication with limited buffers, *Proc. IEEE Symposium on Foundations of Computer Science*, 1984, pp.127-136.

[RANA87] Ranade, A. G., How to emulate shared memory, *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, 1987, pp. 185-194.

[RIVE87] Rivest, Ron, Lecture Notes, MIT, 1987.

[UPFA84] Upfal, E., Efficient schemes for parallel ommunication, *Journal of the ACM*, vol.31, no.3, 1984, pp. 348-355.

[VALI81] Valiant, L. G., and Brebner, G. J., Universal schemes for parallel communication, *Proc. ACM Symposium on Theory of Computing*, 1981, pp. 263-277.

[VALI82] Valiant, L. G., A scheme for fast parallel communication, *SIAM Journal of Computing*, vol.11, no.2, 1982, pp. 350-361.

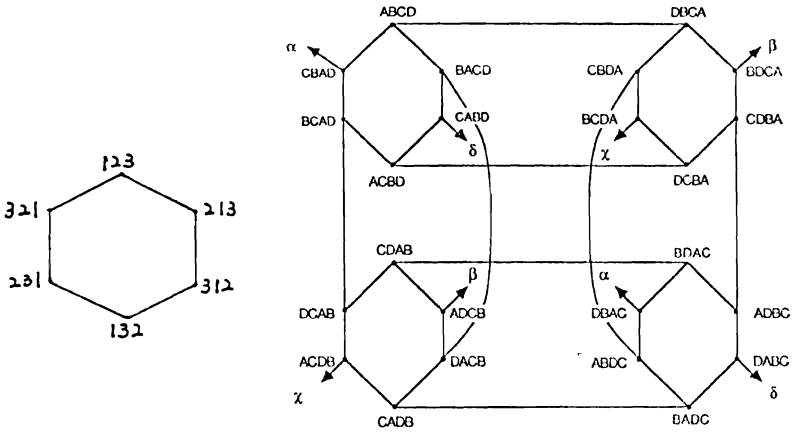

Figure 1: (a) The 3-star graph. (b) The 4-star graph.

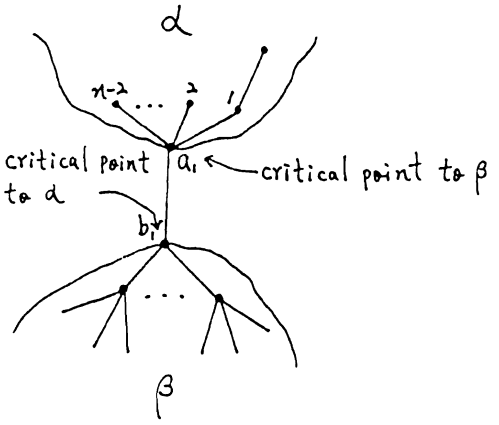

Figure 2:

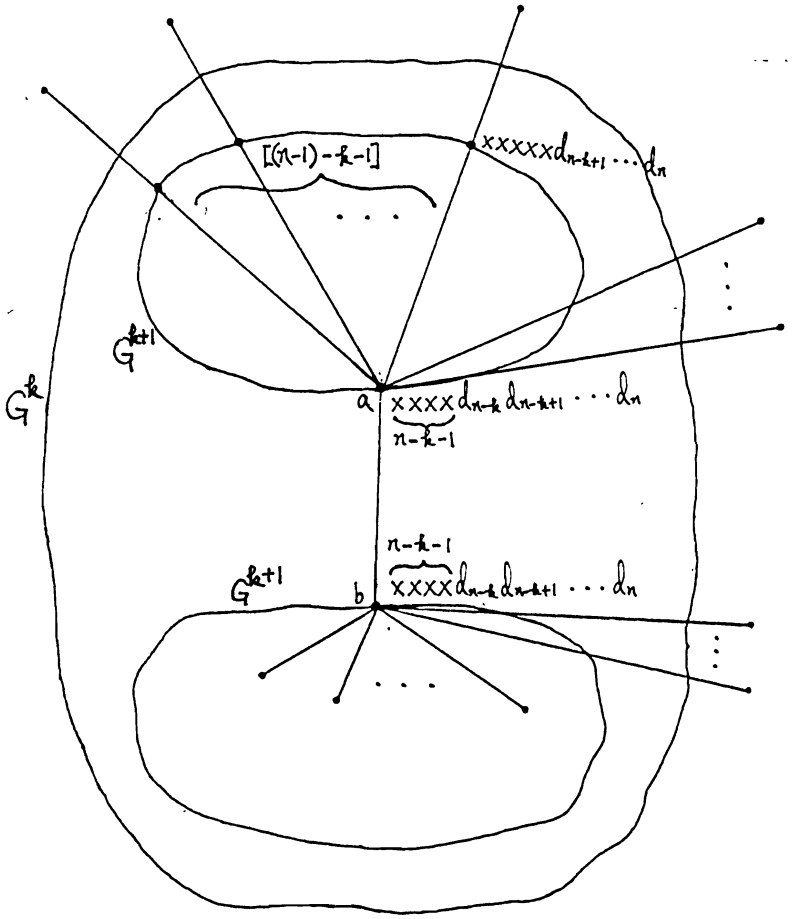

Figure 3:

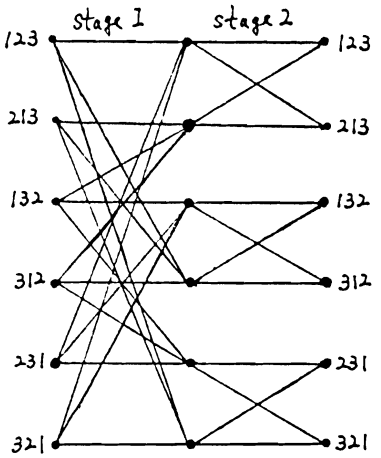

Figure 4: A logical network for the 3-star graph