Fast Algorithms for Bit-Serial Routing on a Hypercube

William A. Aiello[1]
F. T. Leighton[2]
Bruce M. Maggs[3]
Mark Newman[4]

[1]Bell Communications Research
Morristown, New Jersey 07960

[2]Mathematics Department and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

[3]NEC Research Institute
Princeton, NJ 08540

[4]Temple Barker and Sloane Inc.
Lexington, MA 02173

Abstract

In this paper, we describe an $O(\log N)$-bit-step randomized algorithm for bit-serial message routing on a hypercube. The result is asymptotically optimal, and improves upon the best previously known algorithms by a logarithmic factor. The result also solves the problem of on-line circuit switching in an $O(1)$-dilated hypercube (i.e., the problem of establishing edge-disjoint paths between the nodes of the dilated hypercube for any one-to-one mapping).

Our algorithm is adaptive and we show that this is necessary to achieve the logarithmic speedup. We generalize the Borodin-Hopcroft lower bound on oblivious routing by proving that any randomized oblivious algorithm on a polylogarithmic degree network requires at least $\Omega(\log^2 N/\log\log N)$ bit steps with high probability for almost all permutations.

1 Introduction

Substantial effort has been devoted to the study of store-and-forward packet routing algorithms for hypercubic networks. The fastest algorithms are randomized, and can solve any one-to-one packet routing problem on an N-node hypercube in $O(\log N)$ packet steps with high probability [3, 14, 18, 19, 22, 23, 24, 26, 27]. Of the deterministic algorithms [5, 9, 10], the fastest requires $O(\log N \log\log N \log^* N)$ packet steps [9]. One drawback to these algorithms, however, is that they all require $\Omega(\log^2 N)$ bit steps when implemented on real parallel machines such as the Connection Machine. The reason is that most parallel machines operate in a bit-serial or byte-serial fashion because of pin limitations on switches. Since packets always contain at least $\log N$ bits of addressing information, a typical packet step really consists of $\Omega(\log N)$ bit steps.

In general, fast bit-serial algorithms are harder to design than store-and-forward algorithms. This is particularly true on a hypercube since the head of a $\log N$-bit packet should be close to its destination before the tail of the packet leaves the origin[1]. Moreover, while the packet is spread out and

This research was supported by the Defense Advanced Research Projects Agency under Contracts N00014–87–K–825 and N00014–89–J–1988, the Air Force under Contract AFOSR–89–0271, and the Army under Contract DAAL–03–86–K-0171.

[1]Throughout this paper, $\log N$ refers to $\log_2 N$.

moving through the network, it prevents up to $\log N$ wires from being used by other packets. This phenomenon raises issues such as deadlock and livelock [11] that do not arise in the simpler store-and-forward model.

In this paper, we describe an asymptotically optimal randomized algorithm for packet routing on the hypercube. The algorithm solves any permutation routing problem for packets that are M bits long (not including addressing information) in $O(M + \log N)$ bit steps with high probability. Moreover, the algorithm requires each node of the hypercube to deal with only $O(1)$ new packets at each step. The algorithm works in either the wormhole [11] or cut-through [16] models of routing.

The algorithm also solves the problem of on-line circuit switching in an $O(1)$-dilated hypercube, where a *b-dilated* hypercube is one in which each edge is replaced by a bundle of b edges. Previously, it was known that for any one-to-one routing problem, edge-disjoint paths could be established from the inputs to the outputs of a Benes network [6]. The Benes network can be embedded in a 4-dilated hypercube. However, the only on-line algorithms previously known for finding the paths in a Benes network or hypercube require $\Omega(\log^3 N)$ steps [8, 21]. Our algorithm finds a set of edge-disjoint paths on the $O(1)$-dilated hypercube, on-line, in $O(\log N)$ bit steps.

The algorithm has three essential features: (1) it uses Valiant's paradigm of routing to random intermediate destinations, (2) it uses a new embedding of a $\log N$-dilated butterfly into the hypercube that is based on 1-error correcting codes, and (3) it implements a bit-serial version of Ranade's algorithm [18, 19, 23] that eliminates the ghost packets and the need for packets to carry $O(\log \log N)$-bit ranks.

Most hypercube routing algorithms [15, 18, 19, 23, 26, 27] are oblivious. A *deterministic oblivious* routing algorithm is one in which the path taken by a packet through the network is a function of the origin and destination of the packet. Deterministic oblivious algorithms have previously been called oblivious algorithms [7]. A *randomized oblivious* algorithm is one in which each packet independently chooses its path according to a probability distribution which is a function of its origin and destination.

The algorithms described in this paper are not oblivious. In fact, we show that any randomized oblivious algorithm on any N-node network with maximum node degree d requires $\Omega((M + \log N/\log d)\log N/(\log d + \log\log N))$ bit steps, with high probability, for almost all permutations, where M is the packet size (not including addressing information). For routing $\log N$-bit

messages on the hypercube, the lower bound is $\Omega\left(\frac{\log^2 N}{\log\log N}\right)$ bit steps. This result extends the Borodin-Hopcroft $\sqrt{N/d^3}$-packet-step lower bound for deterministic oblivious algorithms [7] to randomized oblivious algorithms. The lower bound for deterministic oblivious algorithms has recently been improved to $\Omega(\sqrt{N}/d)$ by Kaklamanis, Krizanc, and Tsantilas [15], who also provide a deterministic oblivious store-and-forward algorithm for routing on the hypercube in $O(\sqrt{N}/\log N)$ steps. Further work on bounds for oblivious routing and trade-offs between time and randomness can be found in [17].

Although our algorithm is the first route permutations on the hypercube in $O(\log N)$ bit steps with high probability, Leighton and Plaxton [20] have subsequently achieved a stronger result. They show how to sort on bounded-degree hypercubic networks in $O(\log N)$ steps (with the caveat that with very small probability the output will not be in sorted order). Their algorithm can be pipelined and hence can achieve permutation routing as well as many-to-one and one-to-many routing on hypercubic networks in $O(\log N)$ bit steps with high probability, using standard reductions of routing to sorting (plus a minor fix to take care of the case when the output is not sorted). Nonetheless, techniques used in this paper have recently been extended by Aiello and Leighton [1] to construct improved embeddings of trees and other structures in a hypercube and to design more efficient and robust algorithms for reconfiguring a hypercube around random or worst-case faults.

A natural open problem is to find a deterministic algorithm for routing (or sorting) in $O(\log N)$ bit steps on a hypercubic network. The only known bounded-degree networks capable of deterministic $O(\log N)$-bit-step routing are the AKS sorting circuit [2] and the multibutterfly [4, 25]. However, these sorting and routing algorithms rely on expansion properties that the hypercube and its derivatives do not possess.

The remainder of the paper is divided into sections as follows. We start by describing two $O(\log N)$ bit-step algorithms for routing on a $\log N$-dilated butterfly in Section 2. The first is quite simple, but needs a switch to handle up to $\log N$ new packets at once. The second is more efficient, allowing a switch to handle at most 2 new packets at each step. In Section 3, we show how to implement the algorithms on a hypercube by embedding the $\log N$-dilated butterfly in the hypercube. The lower bound for randomized oblivious routing is described in Section 4.

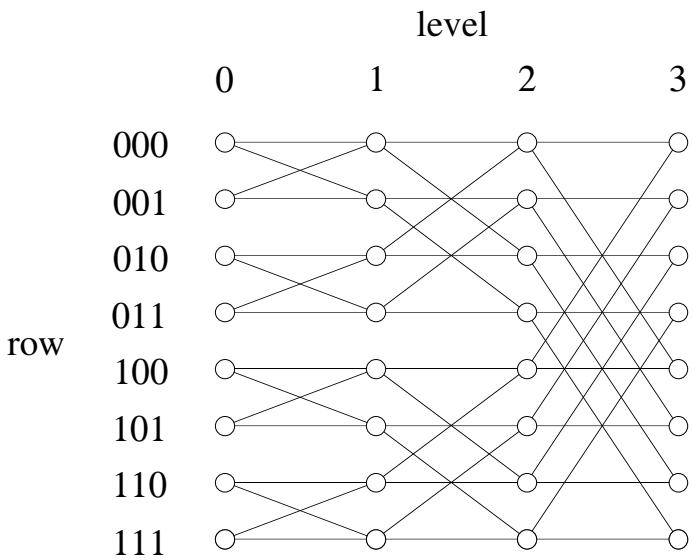

Figure 1: An 8-input butterfly network.

2 Routing on a dilated butterfly

In this section, we describe an algorithm for routing N packets of length M in $O(M + \log N)$ bit steps on an N-node $O(\log N)$-dilated butterfly network. In Section 3, we show that the algorithm can be emulated by an N-node hypercube with constant slowdown.

An 8-input *butterfly network* is illustrated in Figure 1. Each node has a distinct label (r, l), where r is the row, and l is the level. On a butterfly with 2^n inputs, the row is an n-bit binary number and the level is an integer between 0 and n. The nodes on level 0 and n are called the *inputs* and *outputs*, respectively. We will assume that the input and output nodes in each row are identified as the same node, so that the total number of nodes is $N = n2^n$. For $l < n$, a node labeled (r, l) is connected to nodes $(r, l + 1)$ and $(r^{l+1}, l + 1)$, where r^{l+1} denotes r with bit $l + 1$ complemented (bit 1 is the least significant, bit n the most). We will assume that the edges are directed in the order of increasing level (from left to right in Figure 1). We call the edges into a switch its *input edges*, and those out of the switch its *output edges*.

Between any input and output of the butterfly there is a unique path, and there is a simple greedy algorithm for finding the path. Upon reaching switch (r, l), $l < n$, a packet with destination (R, n) compares r^{l+1} with R^{l+1}. If they are equal, the packet takes the edge to $(r, l+1)$. If not, it takes the edge to $(r^{l+1}, l+1)$.

A *b-dilated* butterfly is derived from a butterfly by replacing each edge in the butterfly with a bundle of b edges. A switch in a b-dilated butterfly has 2 input bundles of b input edges each, and 2 output bundles of b output edges each.

There are two models for the switches. In the *strong switch model*, a switch can examine all of its edges at each step. In the *weak switch model*, a switch can examine only one edge from each input bundle at each step. In both models a switch can shunt an input edge to an output edge so that, in the future, a bit received on the input edge is transmitted on the output edge one bit step later, without being examined by the switch. In the strong model, a switch can shunt any number of input-output pairs together in one step; in the weak model, only one. In either model, an edge can transmit at most one bit at each step.

Our goal is to route permutations on a fully loaded butterfly. It is easy to reduce this to the case where each input is the origin of n packets and each output is the destination of n packets. Paths for the packets are selected using Valiant's paradigm [26, 27]; each packet is first routed to a random intermediate destination, and then routed to its true destination. The routing is performed in two phases. In the first phase, each packet is routed from level 0 to a random output on level n. In the second phase, each packet is routed from level 0 to its true destination on level n. (Recall that the inputs and outputs in each row are identified as the same node.) As we shall see, routing through random intermediate destinations ensures that with high probability at most $O(\log N)$ packets pass through any switch in each phase. The network is dilated by this same factor of $O(\log N)$ so that at most one packet ever uses any edge. As a consequence, the routing algorithm can be used for circuit switching as well.

Packets are routed through the network in a *wormhole* fashion [11]. At the end of each edge is a queue that can hold a small number of bits (typically 2). Since a packet carries at least n bits of addressing information, it cannot be stored entirely on one edge, but must be spread out over many edges. A packet can be thought of as a worm working its way head first through the

network. Behind the head, each bit of the worm can advance only if there is space in the queue at the end of the next edge. When the head moves, the queue space it frees up trickles back to the tail, allowing the entire worm to move. When we speak of the location of a packet we refer to its head. Our algorithms also work for arbitrary *cut-through* routing [16] where the entire packet can pile up at one node, provided that there is sufficient queueing at the edges.

At the head of each packet is a $2n$-bit destination header consisting of the row number of its intermediate destination followed by the row number of its final destination. Each time a packet passes through a switch, the lead bit in its header is stripped off and examined to determine which output bundle to send the packet through. The switch then shunts the edge on which the packet has just arrived to a previously unused edge in the outgoing bundle specified by the bit. Such an unused edge can always be found, provided that the number of wires in each outgoing bundle is at least as large as the number of packets that pass through the bundle.

2.1 Routing in the strong switch model

A simple calculation, performed below, reveals that, with high probability, at most $O(\log N)$ packets pass through any bundle. If the butterfly is dilated by this $O(\log N)$ factor, then there will be an unused outgoing edge for each packet that enters a switch. Thus, with high probability, no packet is ever delayed in the strong switch model. At each step, a switch simply examines all of its incoming edges, and shunts those with new packets to outgoing edges.

For simplicity, we will examine the first routing phase only. The analysis of the second phase is similar. Consider a bundle connecting a switch on level l to a switch on level $l+1$. This bundle can be reached from 2^l inputs, and from this bundle $2^{n-(l+1)}$ outputs can be reached. Since there is a unique path from any input to any output, the probability that a packet originating at one of these 2^l inputs passes through the switch is simply the probability that it chooses one of these $2^{n-(l+1)}$ outputs as its random intermediate destination, $2^{n-(l+1)}/2^n = 1/2^{l+1}$. Since n packets originate at each input, and each of these packets chooses its intermediate destination independently, the probability that more than $c \log N$ packets pass through the bundle is at

most $\binom{n2^l}{c \log N}(1/2^{l+1})^{c \log N}$. Using the fact that $\binom{a}{b} \leq (ae/b)^b$ for $0 < b \leq a$, we see that for any constant k, there is a constant c such that this probability is at most $1/N^k$. Summing the probabilities for all $2N$ bundles, we see that the probability that more than $c \log N$ packets pass through any switch is at most $2/N^{k-1}$.

2.2 Routing in the weak switch model

Routing is more complicated in the weak switch model. A problem arises when several packets arrive on the same input bundle at the same step. Since a switch can examine only one edge in the bundle, it can forward only one of them, while the others must wait. The routing algorithm must guarantee that the total delay experienced by each packet is at most $O(\log N)$ bit steps.

Our routing algorithm is a variant of Ranade's algorithm for routing on the butterfly [18, 19, 23]. In Ranade's algorithm, each packet is initially assigned an $O(\log \log N)$-bit random rank, which it carries in front of its destination header. The most important invariant maintained by the algorithm is that the stream of packets leaving each switch is arranged in order of nondecreasing rank. In order to keep the stream sorted, a switch never sends a packet until it is sure that no other packets with smaller rank will arrive later.

The problem with implementing Ranade's algorithm in the bit model is that it takes too long to compare the ranks of two packets at the same switch. In our algorithm, each packet is also assigned a random rank initially, and that rank does not change throughout the course of the routing. However, a packet does not carry its rank along with it. Instead, we introduce a new type of packet called a *token*. A token carries no information other than its type, and as a consequence consists of only $O(1)$ bits. The algorithm uses these tokens to represent the ranks of the other packets. It maintains the invariant that a packet of rank r passes through a bundle only after exactly r tokens have passed through the bundle. This invariant guarantees that the packets pass through the bundle in order sorted by rank. A token can be considered to have a rank as well; a token of rank r passes through a bundle only after exactly r tokens have passed through the bundle. Note that the tokens are not the same as the ghost packets used in Ranade's algorithm, which carry a rank. Our algorithm does not use ghost packets.

The mechanics of the algorithm are fairly simple. A switch maintains a pointer to an edge in each of its two input and two output bundles. We call these edges the *current* input and output edges, respectively. The behavior of the switch depends on whether the current input queues contain packets or tokens. There are four cases. If both of the current input edges have tokens in their queues, then the switch sends tokens on both of the current output edges and advances all four pointers. If one of the current input edges has a token in its queue and the other a packet, then the switch strips off the first bit of the packet's header, shunts the packet to the current edge of the output bundle specified by that bit, and advances the pointers for the input and output bundles used by the packet. If both of the current input edges have packets in their queues, then the switch forwards one of them, and advances the pointers for the bundles used by that one. If either of the current input edges has an empty queue, then the switch does nothing.

Initially, each packet p is assigned a random rank $\text{rank}(p)$ in the range $[0, \ldots, w-1]$, where $w = O(\log N)$, and will be specified later. The input switches on level 0 send out a stream of packets interspersed with tokens so that r tokens precede each packet with rank r. A total of w tokens are sent on each bundle. The tokens with rank $w-1$ follow all of the other packets and tokens in the network. A routing phase terminates when every output has received a rank $w-1$ token on both of its input bundles. Rank $w-1$ tokens can be specially marked so that the outputs do not have to count the number of tokens that they have received.

During the course of the algorithm, each edge is used by either one packet, one token, or not at all. Because w tokens pass through each bundle, we further dilate the butterfly by adding $w = O(\log N)$ edges to each bundle. By using the same edges to send both packets and tokens, it is possible to remove these extra w edges. However, the algorithm and its analysis become slightly more complicated because several tokens may be sent across the same edge, and each one must wait to be sent until a space frees up in the queue at the end of the edge.

The proof that the heads of the packets quickly reach their destinations uses a delay sequence argument like that in [18, 19, 23]. A *k-delay sequence* consists of four components: a path backwards from an output to an input, a sequence of (not necessarily distinct) switches appearing on the path in the order $s_1, \ldots, s_k$, a sequence $p_1, \ldots, p_k$ of distinct packets, and a sequence $r_1, \ldots, r_k$ of non-increasing ranks. We say that a k-delay sequence has *oc-*

curred if packet p_i passes through s_i for $1 \leq i \leq k$, and $\text{rank}(p_i) = r_i$ for $1 \leq i \leq k$.

The following lemma is the crux of the delay sequence argument. It shows that if some packet is delayed, then a delay sequence must have occurred. For simplicity we analyze the first routing phase only.

Lemma 2.1 *For any $\delta > w$, if some packet arrives at its intermediate destination at step $n + \delta$, then a $(\delta - w)$-delay sequence must have occurred.*

Proof: We will construct a delay sequence by spending *lag* points. The lag of a packet on level l at time τ is $\tau - l$. If some packet arrives at its destination at step $n + \delta$, then it has lag δ. We use these δ points to build the sequence. For each point spent, we will either insert a packet into the sequence, or decrease the rank of the next packet in the sequence. Since there can be at most w decreases in the rank, the sequence must contain at least $\delta - w$ packets.

The construction of the delay sequence begins with the last packet to arrive at its destination. Suppose a packet p_1 arrives at its destination s_1 at time $n + \delta$. First, we insert p_1, s_1, and $r_1 = \text{rank}(p_1)$ into the delay sequence. Then we follow p_1 back in time until it was last delayed. The lag remains δ at each step on the path, until the step at which p_1 failed to move forward, where the lag drops to $\delta - 1$.

If p_1 was delayed at switch s_2 because another packet p_2 was sent instead, then $\text{rank}(p_2) = \text{rank}(p_1)$, and we insert p_2, s_2, and $r_2 = \text{rank}(p_2)$ into the sequence, and follow p_2 back in time. In this case, we have inserted one packet into the sequence at a cost of one lag point. In the butterfly network, once the paths of two packets diverge, they do not meet again. Thus, if p_2 delays p_1, then p_2 remains ahead of p_1 as long as their paths intersect. As a consequence, no packet is inserted into the sequence more than once.

If p_1 was delayed because a token t_1 was sent instead, then $\text{rank}(t_1) = \text{rank}(p_1) - 1$. In this case we do not insert anything into the sequence, but instead follow t_1 back in time until it was last delayed. In this case, we have lost one point in lag, but have also decreased the rank of the next packet in the sequence by at least one point.

The third possibility is that p_1 was delayed because the current edge on the other input bundle, c, had nothing to send. In this case we do not insert anything into the sequence, but go back through c to a switch s at the

previous level at the previous time step. At the previous time step, switch s must not have sent anything through bundle c. This can happen for one of two reasons. Either s sent a packet p_2 on its other bundle, or one of the input bundles c' to s had nothing to send. In the former case, we insert p_2, $s_2 = s$, and $r_2 = \text{rank}(p_2) = \text{rank}(p_1)$ into the sequence, and follow p_2 back until it was last delayed. In the latter case, we go back through c' to a switch s' at the previous level at the previous time step and continue. Continuing in this fashion, we must eventually reach a switch that sent a packet.

Now that we have analyzed the ways in a which a packet can be delayed, we must consider the ways in which a token can be delayed. Suppose we are following a token t_1 back in time until it was last delayed at a switch s.

If t_1 was delayed because s sent a packet p instead, then we insert $p_2 = p$, $s_2 = s$, and $r_2 = \text{rank}(p) = \text{rank}(t_1)$ into the sequence.

Otherwise, if t_1 was delayed because s sent another token t_1' instead, then $\text{rank}(t_1') = \text{rank}(t_1) - 1$, and we follow t_1' back until it was last delayed.

The last possibility is that t_1 was delayed because one of the input bundles c to s had nothing to send. In this case, we go back through c just as if a packet had been delayed for the same reason.

The preceding case analysis shows that for each point of lag spent, either a packet is inserted into the delay sequence, or the rank of the next packet in the sequence decreases. Thus, the sequence must contain at least $\delta - w$ packets. □

Theorem 2.2 *For any fixed constant k_1, there is a fixed constant k_2 such that the probability that some packet is delayed in arriving at its intermediate destination for $k_2 \log N$ steps is at most $1/N^{k_1}$.*

Proof: To bound the probability that some packet is delayed for δ steps, we now need only to bound the probability that some $\delta - w$-delay sequence occurs. To do so, we enumerate all possible delay sequences, and sum the probabilities that each occurs. The argument is similar to the ones in [18, 19, 23]. There are 2^{2n} ways to choose the path, since there are 2^n choices for the output at which it starts, 2^n choices for the input at which it ends, and only one path between them. Once the path is fixed, there are at most $\binom{n+\delta-w}{\delta-w}$ ways of choosing switches $s_1, \ldots, s_{\delta-w}$ on the path. Let l_i denote the level of switch s_i. Then there are at most $\prod_{i=1}^{\delta-w}(n2^{l_i})$ ways to choose $p_1, \ldots, p_{\delta-w}$. Finally, there are at most $\binom{\delta}{\delta-w} = \binom{\delta}{w}$ ways of choosing a sequence $r_1 \ldots r_{\delta-w}$

of non-increasing ranks in the range $[0, w-1]$. A delay sequence occurs only if p_i passes through s_i and $\text{rank}(p_i) = r_i$, for $1 \leq i \leq \delta - w$. For the delay sequence constructed thus far, the probabilities that these two events occur are $\prod_{i=1}^{\delta-w}(2^{n-l_i}/2^n)$, and $1/w^{\delta-w}$, respectively. Thus, the probability that any delay sequence occurs is at most

$$\frac{2^{2n}\binom{n+\delta-w}{\delta-w}n^{\delta-w}\binom{\delta}{w}}{w^{\delta-w}}.$$

Using the inequality $\binom{a}{b} \leq (ae/b)^b$ for $0 < b \leq a$, it is straightforward to show that for any k_1, there are constants k_2 and k_3, such that for $\delta = k_2 \log N$, and $w = k_3 \log N$, $k_2 > k_3$, the product is at most $1/N^{k_1}$. □

The analysis for the second phase of routing is similar. If we require the first phase to be completed before the second phase begins, then each takes $O(\log N)$ bit steps, and the total time is $O(\log N)$ bit steps. In fact, it is possible to pipeline the two phases so that some packets begin the second phase before the first is finished. The resulting algorithm is simpler and still requires only $O(\log N)$ bit steps. The analysis is a little more complicated because a packet p_1 can delay another packet p_2 in the first phase, and p_2 can delay p_1 in the second.

Once the head of a packet reaches its true destination, the M bits of data following behind it trickle into the destination in $O(M)$ bit steps. Since, with high probability, all of the heads reach their destinations in $O(\log N)$ bit steps, the total time is also $O(M + \log N)$ with high probability.

3 Emulating the routing algorithm on the hypercube

In this section we will show that an N-node hypercube can emulate the routing algorithm performed by an N-node $O(\log N)$-dilated butterfly with constant slowdown. Our strategy is to break the butterfly into two halves and to embed each half in the hypercube with constant load, congestion, and dilation. The halves are wired together by a set of paths with constant congestion and dilation $O(\log N)$.[2] Although a packet may be delayed for

[2]See [12] for another embedding with the same congestion, load, and dilation.

$O(\log N)$-bit steps while traveling from one half to the next, the total time remains $O(\log N)$.

Our embedding will rely crucially on the notion of a star partition. This in turn will rely on perfect 1-error correcting codes which were first constructed by Hamming [13] for words of $2^r - r - 1$ bits. More specifically, for every word w of length $2^r - r - 1$ there exists a codeword, $c(w)$, of length $2^r - 1$ such that every word of length $2^r - 1$ is either a codeword or has Hamming distance 1 from exactly one codeword.

A hypercube is a network where the number of nodes, N, is a power of two, say $N = 2^s$. The exponent s is called the dimension of the cube. Each node has an s-bit address and an undirected edge to each of the s nodes whose address differs in one bit.

Suppose that we have a hypercube (or subcube) with $s = 2^r - 1$ dimensions. A node, v, is a *star center* if v is a codeword: $v \in \{c(w)|w \in \{0,1\}^{2^r-r-1}\}$. Define a *star* to be a star center and all of its neighbors. The node $c(w)^i$ differs from $c(w)$ in the ith bit and is called the ith *arm* of the $c(w)$ star. Since c is a perfect 1-error correcting code, every node in the hypercube is either a star center or the neighbor of a star center (i.e., the arm of a star). That is, the stars partition the nodes of the hypercube. An edge of a hypercube is either between a star center and an arm or between arms on different stars.

Let $n = 2^k - 1$. For simplicity suppose that the butterfly is $2n - k + 1$ by 2^{2n-k} and that every dilated edge is actually composed of n edges. We will embed this into a hypercube of dimension $2n$. Observe that the hypercube has approximately twice as many nodes. As a first step we will embed levels 0 to n of the butterfly. To do so we will first give an embedding of the butterfly nodes in these levels and then we will give a mapping of the butterfly edges.

A node in the first $n + 1$ levels of the butterfly can be described by the pair $(m \circ l, i)$ where $m \in \{0,1\}^{n-k}$ consists of the most significant bits of the row in which the node resides, $l \in \{0,1\}^n$ consists of the least significant bits, and $0 \leq i \leq n$ is the level in which the node resides. Since any path in the butterfly passes through nodes with the same value of m in the first $n + 1$ levels, all nodes with the same value of m will be considered as the sub-butterfly B_m. Our mapping of nodes in the sub-butterflies to nodes in the hypercube is defined as follows:

$$(m \circ l, i) \mapsto c(m)^i \circ l$$

for $1 \leq i \leq n$. Map $(m \circ l, 0)$ (as well as $(m \circ l, n)$ from above) to $c(m)^n \circ l$.

Note that since m is composed of $n-k$ bits, $c(m)$ is composed of n bits and hence the hypercube has dimension $2n$. Let C_l be the sub-cube where the least significant bits of all the nodes agree with l. Note that all the nodes in row l of each sub-butterfly are mapped to a star in C_l. In fact, C_l is partitioned by the stars formed from the lth row of every sub-butterfly. Consider two rows of B_m that differ by one bit, say l and l^i. One gets mapped to the $c(m)$ star in C_l and the other to the $c(m)$ star in C_{l^i} such that the corresponding arms of each star differ in the ith bit. The map suggests that we can use the ith dimension of the hypercube to route the ith dimension (level) of the butterfly.

Now we must show how to map butterfly edges to hypercube edges. Consider the dilated edge between $(l, i-1)$ and (l^i, i) in the B_m sub-butterfly for $2 \leq i \leq n$ (we will handle $(l,0) \rightarrow (l^1, 1)$ separately). The first node of this edge is the $i-1$st arm of the $c(m)$ star in C_l and the endpoint is the ith arm of the $c(m)$ star in C_{l^i}. We need to distribute the n edges of the dilated edge in B_m to the n edges between the arms of the $c(m)$ stars in the two sub-cubes. Let $(m \circ l, i-1) \rightarrow^j (m \circ l^i, i)$ denote the jth edge of the dilated edge between $(m \circ l, i-1)$ and $(m \circ l^i, i)$ in the butterfly. This edge is mapped to the following path in the hypercube. Starting at $c(m)^{i-1}$ in C_l we first go to $c(m)^j$ via $c(m)^{i-1,j}$. Then we take the edge which takes us to $c(m)^j$ in C_{l^i}. Finally, we go to $c(m)^i$ via $c(m)^{i,j}$ in C_{l^i}. More succinctly,

$$c(m)^{i-1} \circ l \rightarrow c(m)^{i-1,j} \circ l \rightarrow c(m)^j \circ l \rightarrow$$
$$c(m)^j \circ l^i \rightarrow c(m)^{i,j} \circ l^i \rightarrow c(m)^i \circ l^i.$$

For the edge $(l,0) \rightarrow^j (l^1, 1)$ we make the obvious modification:

$$c(m)^n \circ l \rightarrow c(m)^{n,j} \circ l \rightarrow c(m)^j \circ l \rightarrow$$
$$c(m)^j \circ l^1 \rightarrow c(m)^{1,j} \circ l^1 \rightarrow c(m)^1 \circ l^1.$$

A straight edge in the butterfly, $(m \circ l, i-1) \rightarrow^j (m \circ l, i)$, $2 \leq i \leq n$, is mapped in a similar way except that we don't need to change the ith bit of l:

$$c(m)^{i-1} \circ l \rightarrow c(m)^{i-1,j} \circ l \rightarrow c(m)^j \circ l \rightarrow$$
$$c(m)^{i,j} \circ l \rightarrow c(m)^i \circ l$$

for $2 \leq i \leq n$. Again we make the obvious modification for $i = 1$:

$$c(m)^n \circ l \rightarrow c(m)^{n,j} \circ l \rightarrow c(m)^j \circ l \rightarrow$$
$$c(m)^{1,j} \circ l \rightarrow c(m)^1 \circ l.$$

Claim 3.1 *At most 16 butterfly edges are mapped through any hypercube edge by the preceding embedding.*

Proof: From the construction, we know that butterfly edges are routed only through edges that link arms of different stars. There are then two cases to consider, depending on whether or not the stars are in different subcubes.

Case 1: hypercube edges of the form

$$(c(\underline{m})^{\underline{j}} \circ \underline{l}, c(\underline{m})^{\underline{j}} \circ \underline{l}^{\underline{i}}).$$

Such an edge must be the third hypercube edge used for a crossing butterfly edge. Moreover, the butterfly edge must be of the form $(m \circ l, i-1) \rightarrow^j (m \circ l^i, i)$ where $i = \underline{i}$, $l = \underline{l}$ or $l = \underline{l}^{\underline{i}}$, $j = \underline{j}$ and $m = \underline{m}$. Hence, this hypercube edge is used by at most two edges from the dilated butterfly.

Case 2: hypercube edges of the form

$$(c(\underline{m})^{\underline{i}} \circ \underline{l}, c(\underline{m})^{\underline{i},\underline{j}} \circ \underline{l}).$$

Such an edge can be the first, second, fourth, or fifth edge in a path for $(m \circ l, i-1) \rightarrow^j (m \circ l^i, i)$, or any of the four edges in a path for $(m \circ l, i-1) \rightarrow^j (m \circ l, i)$. In each case, there are two possible choices for the values of m, l, i, and j, depending on the direction in which the edge is used. Hence, each such edge is used at most 16 times. □

The remaining levels of the dilated butterfly can be mapped to the hypercube in an analogous fashion. In particular, we can map the last $n+1$ levels of the butterfly to the hypercube by using the same algorithm as before, except that butterfly node $(m' \circ l', i')$ is mapped to hypercube node $c(l')^{i'-n+k} \circ m$, where $m' \in \{0,1\}^n$, $l' \in \{0,1\}^{n-k}$, and $n - k \leq i' \leq 2n - k + 1$. (Note that we simply swapped the roles of the most and least significant bits.) Butterfly edges are mapped in an analogous fashion.

Unfortunately, the embedding process just described maps each node in the middle $k+1$ levels of the butterfly to two different places. We will resolve

this problem by using only the location prescribed by the first embedding for nodes on levels $n-k$ through n.

We are left with the problem of embedding the edges from level n to level $n+1$, since the nodes on level n are located by the embedding of the first $n+1$ levels of the butterfly, and the nodes on level $n+1$ are located by the embedding of the last $n+1$ levels of the butterfly. We do not know how to embed these edges with constant dilation, but we can embed them with constant congestion and $O(\log N)$ dilation as follows. Recall that the nodes on level n of the butterfly are mapped one-to-one to the nth arms of stars, while the nodes of level $n+1$ are mapped one-to-one to the $k+1$st arms of stars. The path for the jth edge between level n and $n+1$ begins (ends) with a trip to (from) the jth arm of a star. This can be done with $O(1)$ congestion and dilation 2 for both the level n and level $n+1$ nodes using the paths

$$c(m)^n \circ l \rightarrow c(m)^{n,j} \circ l \rightarrow c(m)^j \circ l,$$

or

$$c(m)^{k+1} \circ l \rightarrow c(m)^{k+1,j} \circ l \rightarrow c(m)^j \circ l.$$

Since each of these star arms are distinct, the paths can be completed with $O(1)$ congestion and $O(\log N)$ dilation using Benes routing [6].

The hypercube can now easily emulate the butterfly routing algorithms from Section 2. Each hypercube node emulates at most 2 butterfly nodes. Also, since the first n levels of butterfly edges are embedded with constant congestion and dilation, the hypercube can emulate transmission along these edges with constant slowdown. The same is true of the last $n-k-1$ levels of butterfly edges. Only the edges from level n to $n+1$ cause problems. Since these edges are embedded with dilation $O(\log N)$, it takes $O(\log N)$ bit steps to emulate transmission along them. Fortunately, the routing algorithms use the butterfly edges in order of increasing dimension from 0 to $2n-k-1$, so the $O(\log N)$-step delay experienced along the edges from level n to $n+1$ is additive.

4 Lower bounds on oblivious routing

In this section we prove that any randomized oblivious routing algorithm takes

$$\Omega\left(\frac{\left(M+\frac{\log N}{\log d}\right)\log N}{\log d+\log\log N}\right)$$

bit steps with high probability for almost all permutations, where M is the minimum packet size not including addressing information, N is the size of the network and d is the maximum degree of the network. The proof is similar to the classic Borodin-Hopcroft lower bound proof [7] but extends their method and result in two ways. First, the lower bound applies to almost all permutations, and second, it applies to a wider class of algorithms: randomized oblivious algorithms as opposed to deterministic oblivious algorithms. Our only assumption is that each packet has at least Δ bits of addressing information if it is still a distance Δ away from its destination. Without this assumption, the lower bound is $\Omega(M\log N/(\log d+\log\log N))$ bit steps.

Let us first define the notion of a randomized oblivious routing algorithm. We shall assume that our network is a directed graph $G=(V,E)$, where $|V|=N$ and the in-degree and out-degree of every node is at most d. Let p denote a sequence of edges which form a path (not necessarily simple) in the graph; let $P_{u,v}$ be the set of all paths from u to v. For a given randomized routing algorithm let $\mathcal{P}_{u,v}$ be the random variable denoting the path from u to v chosen by the algorithm. Using this notation, for any node or edge x and for all nodes u and v we have the following:

$$\sum_{p\in P_{u,v}}\Pr[\mathcal{P}_{u,v}=p]=1,$$

and

$$\Pr[x\in\mathcal{P}_{u,v}]=\sum_{p:\ x\in p}\Pr[\mathcal{P}_{u,v}=p].$$

A *randomized oblivious* routing algorithm is one in which all distinct random variables $\mathcal{P}_{u,v}$ are independent.

We should discuss how the definition of obliviousness applies to networks which have multiple edges. We consider each edge of a bundle of multiple edges to be distinct so that paths which use different edges in the same bundle are considered to be different. This implies that oblivious algorithms

on networks with multiple edges must be oblivious to the choice of edges within bundles of edges. Hence, even though our routing algorithm for the dilated butterfly chooses the nodes along a packet's path obliviously, it is not an oblivious algorithm because the choices of edges within a bundle depended upon the choices made by other packets. This is what allows us to achieve $O(\log N)$ bit steps and beat the lower bound.

Networks where edges have capacity c (and degree d') are considered to be networks with multiple edges and degree $d = cd'$. Hence, for an N node hypercube with capacity $O(1)$ on its edges, i.e., $d = O(\log N)$, we find that any randomized oblivious algorithm takes $\Omega\left((\frac{\log N}{\log\log N})^2\right)$ bit steps with high probability, even if $M = O(1)$. In fact, messages typically have length $M = \Omega(\log N)$. For $M = \Theta(\log N)$, the bound is $\Omega(\left(\log^2 N/\log\log N\right)$. More generally, if $M = \Omega(\log N/\log\log N)$, the bound is $\Omega\left(M \log N/\log\log N\right)$.

Given a randomized oblivious routing algorithm for a network we would like a measure of how likely it is for a node or edge to be in a randomly selected set of paths. For any node v, any set of nodes T, and any node or edge x, we define the *weight* of x with respect to v and T to be

$$W_v^T(x) = \sum_{t \in T} \Pr[x \in \mathcal{P}_{v,t}].$$

Note that the weight of a node is equal to the sum of the weights of its incoming (or outgoing) edges.

Observe that if we choose a random destination $t \in T$ and a random path from v to t according to the $\mathcal{P}_{v,t}$, then x will be contained in the path with probability $\frac{W_v^T(x)}{|T|}$. Similarly, if we choose a random path from v to a randomly selected destination in V, then x will be contained in the path with probability at least $\frac{W_v^T(x)}{N}$.

The goal of the proof is to show that there are many edges e that have many nodes v for which $W_v^T(e)$ is large for some T. Then we will be able to argue that many edges have at least some chance of having many packets pass through them. (We will also argue that these packets still have a long way to go after using the edge in order to show that many bits are likely to pass through the edge.) Then we will conclude that at least one of these edges has high congestion with very high probability.

We start by proving that for any node v and any large set T, there are many pairs $u \in V, T' \subset T$ for which $W_v^{T'}(u)$ is large and u is far away

from T'. Henceforth, we use the notation $\mathrm{dist}(x, T)$ to denote the minimum distance between x and any element of T in the graph.

Lemma 4.1 *For any set of at least $N/2$ nodes T, and any node v, there are at least $\sqrt{N}/2\sqrt{d}$ nodes u for which there exists some $T_{v,u} \subseteq T$ such that $W_v^{T_{v,u}}(u) \geq \sqrt{N}/4\sqrt{d}$ and $\mathrm{dist}(u, T_{v,u}) \geq \Omega(\log N/\log d)$.*

Proof: Consider the set of nodes $U = \{u : W_v^T(u) \geq k\}$ where $k = \sqrt{N}/2\sqrt{d}$. Observe that $N/2$ is a lower bound for $W_v^T(v)$, since $|T| \geq N/2$ and $\Pr[v \in \mathcal{P}_{v,t}] = 1$ for all t. Now we will derive an upper bound for $W_v^T(v)$ in terms of $|U|$. This in turn will imply a lower bound for $|U|$. By definition,

$$\begin{aligned} W_v^T(v) &= \sum_{t \in T} \sum_{p \in P_{v,t}} \Pr[\mathcal{P}_{v,t} = p] \\ &\leq |U| + \sum_{t \in T-U} \sum_{p \in P_{v,t}} \Pr[\mathcal{P}_{v,t} = p]. \end{aligned}$$

Observe that $v \in U$. For any path in the preceding sum from v to a node in $T - U$ let y be the first node not in U and let Y be the set of all y. We can partition the paths according to Y as follows:

$$\sum_{y \in Y} \sum_{t \in T-U} \sum_{\substack{p \in P_{v,t} \\ y \in p}} \Pr[\mathcal{P}_{v,t} = p]$$

$$\leq \sum_{y \in Y} W_v^T(y).$$

This is bounded by $\leq dk\,|U|$ since the size of Y is at most $d|U|$ and the weight of any node not in U is at most k. Thus,

$$W_v^T(v) \leq (1 + dk)\,|U|\,.$$

Combining the two inequalities for $W_v^T(v)$ reveals that $|U| \geq N/2(1 + dk) \geq \sqrt{N}/2\sqrt{d}$.

For each $u \in U$, define $T_{v,u}$ to be the subset of nodes in T that are not within distance $(\log(\sqrt{N}/8\sqrt{d}))/\log d = \Theta(\log N/\log d)$ of u. Since the maximum node degree is d, there are at most $2d^h$ nodes within distance h of a node. Hence, there are at most $\sqrt{N}/4\sqrt{d}$ nodes within distance

$(\log(\sqrt{N}/8\sqrt{d}))/\log d$ of u, and thus

$$\begin{aligned} W_v^{T_{v,u}}(u) &\geq W_v^T(u) - \frac{\sqrt{N}}{4\sqrt{d}} \\ &\geq \frac{\sqrt{N}}{4\sqrt{d}}. \end{aligned}$$

□

The above lemma allows us to show that there is an edge with large weight.

Lemma 4.2 *For any two sets S and T of at least $N/2$ nodes there is an edge e such that there are at least $\sqrt{N}/4d^{3/2}$ nodes $v \in S$ with associated subsets $T_{v,e} \subseteq T$ such that*

$$W_v^{T_{v,e}}(e) \geq \frac{\sqrt{N}}{4d^{3/2}}$$

and $\text{dist}(e, T_{v,e}) \geq \Omega(\log N/\log d)$.

Proof: By Lemma 4.1 and the fact that $|S| \geq N/2$, we know that there are $N^{3/2}/4\sqrt{d}$ pairs of nodes v, u, where $v \in S$, and each pair has an associated subset $T_{v,u} \subset T$ such that $W_v^{T_{v,u}}(u) \geq \sqrt{N}/4\sqrt{d}$ and $\text{dist}(u, T_{u,v}) \geq \Omega(\log N/\log d)$. Recall that the weight of a node is equal to the sum of the weights of its incoming edges and that every node has maximum in-degree d. Hence, there are at least $N^{3/2}/4\sqrt{d}$ node-edge pairs v, e for which $W_v^{T_{v,e}}(e) \geq \sqrt{N}/4d^{3/2}$, $v \in S$, $T_{v,e} \subseteq T$, and $\text{dist}(e, T_{v,e}) \geq \Omega(\log N/\log d)$. Since there are at most dN edges, we can therefore conclude that there is at least one edge e for which there are at least $\sqrt{N}/4d^{3/2}$ nodes $v \in S$ for which $W_v^{T_{v,e}}(e) \geq \sqrt{N}/4d^{3/2}$, $T_{v,e} \subseteq T$, and $\text{dist}(e, T_{v,e}) \geq \Omega(\log N/\log d)$. □

We are now ready to prove the lower bound on randomized oblivious algorithms.

Theorem 4.3 *Every randomized oblivious routing algorithm will take*

$$\Omega((M + \log N/\log d)\log N/(\log d + \log\log N))$$

bit steps with probability at least $1 - e^{-N^{1/4}}$ to route all but an $e^{-N^{1/4}}$ fraction of the $N!$ possible permutations.

Proof: We will show that for a random permutation, the probability that every edge has fewer than $c \log N/(\log d + \log\log N)$ packets passing through it that are $\Omega(\log N/\log d)$ steps away from their destination is at most $e^{-2N^{1/4}}$ for a sufficiently small constant c. In this case, the probability is taken over the choice of permutations as well as the choice of paths. This directly implies the theorem since if more than an $e^{-N^{1/4}}$ fraction of the possible permutations finished in fewer than $O((M + (\log N/\log d))c \log N/(\log d + \log\log N))$ bit steps with probability exceeding $e^{-N^{1/4}}$, then we would have more than a $e^{-2N^{1/4}}$ chance of finishing in fewer steps with a random permutation which is not possible.

We will construct the random permutation by determining one path at a time. We will randomly select a path for each node v in a specified sequence, selecting paths from the distributions for $\mathcal{P}_{v,u}$ where u is randomly selected from the nodes not already designated to receive a packet. (The constraint of constructing a random *permutation* rather than a random destination requires that at most one packet be delivered to any node. The argument for random destinations is somewhat easier.)

We start by selecting an edge e and $m = \sqrt{N}/8d^{3/2}$ nodes $v_1, v_2, \ldots, v_m \in S$, where $S = T = V$, such that for $1 \leq i \leq m$,

$$W_{v_i}^{T_{v_i,e}}(e) \geq \frac{\sqrt{N}}{4d^{3/2}}$$

where $T_{v_i,e} \subseteq T$ and $\mathrm{dist}(e, T_{v_i,e}) \geq \Omega(\log N/\log d)$. Such an edge and set of nodes is guaranteed to exist by Lemma 4.2. We first select a random destination t_1 from $T_{v_1,e}$ and a random path p_1 from P_{v_1,t_1} for v_1. Since $W_{v_1}^{T_{v_1,e}}(e) \geq \sqrt{N}/4d^{3/2}$, we know that the path p_1 contains e and continues afterward for at least $\Omega(\log N/\log d)$ steps with probability at least $1/4d^{3/2}\sqrt{N}$.

We next select a random destination t_2 from $T_{v_2,e} - t_1$ and a random path p_2 from P_{v_2,t_2} for v_2. Since

$$\begin{aligned} W_{v_2}^{T_{v_2,e}-t_1}(e) &\geq W_{v_2}^{T_{v_2,e}}(e) - 1 \\ &\geq \frac{\sqrt{N}}{4d^{3/2}} - 1 \\ &\geq \frac{\sqrt{N}}{8d^{3/2}}, \end{aligned}$$

we know that p_2 uses e and continues afterward for at least $\Omega(\log N/\log d)$ steps with probability at least $1/8d^{3/2}\sqrt{N}$. Moreover, we know that this

probability is independent of the probability that p_1 uses e and continues afterward for $\Omega(\log N/\log d)$ steps.

Arguing in a similar fashion, we can show that $p_1, \ldots, p_m$ each use e and continue afterward for at least $\Omega(\log N/\log d)$ steps with probability $1/8d^{3/2}\sqrt{N}$, and that these probabilities are independent. Since there are $\sqrt{N}/8d^{3/2}$ paths, each using e and continuing afterwards for $\Omega(\log N/\log d)$ steps with probability $1/8d^{3/2}\sqrt{N}$, the probability that at least

$$r = c\log N/(3\log d + \log\log N)$$

paths use e and continue afterwards for $\Omega(\log N/\log d)$ steps is at least

$$\alpha\left(\frac{1}{64d^3 r}\right)^r$$

for some constant α and sufficiently large N. By selecting c to be a small enough constant, this probability can be made to be at least $N^{-1/4}$. In other words, at least $\Omega((M + \log N/\log d)\log N/(\log d + \log\log N))$ bits pass through e from packets starting at nodes $v_1, \ldots, v_m$ with probability at least $N^{-1/4}$.

We now start over again, setting $S = V - \{v_1, \ldots, v_m\}$ and $T = V - \{t_1, \ldots, t_m\}$. Since $m = \sqrt{N}/8d^{3/2}$, $|S| = |T| \geq N/2$, and we can apply Lemma 4.2 to find an edge e' and nodes $v'_1, \ldots, v'_m \in S$ such that for $1 \leq i \leq m$

$$W_{v'_i}^{T_{v'_i,e'}}(e') \geq \frac{\sqrt{N}}{4d^{3/2}}$$

where $T_{v'_i,e'} \subseteq T$ and $\mathrm{dist}(e', T_{v'_i,e'}) \geq \Omega(\log N/\log d)$. Arguing just as before (except that now t'_i is selected uniformly from $T_{v'_i,e'} - \{t'_1, \ldots, t'_{i-1}\}$) we can prove that at least

$$\Omega((M + \log N/\log d)\log N/(\log d + \log\log N))$$

bits pass through e' from packets starting at nodes $v'_1, \ldots, v'_m$ with probability at least $N^{-1/4}$.

We can repeat this whole process $(N/2)/m = 4d^{3/2}\sqrt{N}$ times before violating our condition that S and T have more than $N/2$ nodes. Each repetition we select destinations and paths for m more nodes. Each time we severely congest an edge with probability at least $N^{-1/4}$. Since these

probabilities are all independent, the probability that none of the edges is severely congested is at most

$$\begin{aligned}(1 - N^{-1/4})^{4d^{3/2}\sqrt{N}} &\leq e^{-4d^{3/2}N^{1/4}} \\ &\leq e^{-2N^{1/4}}.\end{aligned}$$

Hence for a random permutation and random path selection, the running time will be $\Omega((M + \log N/\log d)\log N/(\log d + \log\log N))$ with probability $1 - e^{-2N^{1/4}}$. □

5 Acknowledgments

The authors thank Greg Plaxton for many helpful comments.

References

[1] W. Aiello and T. Leighton. Coding theory, hypercube embeddings, and fault tolerance. Unpublished manuscript, 1991.

[2] M. Ajtai, J. Komlos, and E. Szemeredi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[3] R. Aleliunas. Randomized parallel communication. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 60–72, August 1982.

[4] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a non-blocking network. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 149–158, May 1990.

[5] K. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computing Conference*, volume 32, pages 307–314, 1968.

[6] V. E. Benes. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, 1965.

[7] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130–145, February 1985.

[8] R. E. Cypher. *Efficient Communication in Massively Parallel Computers*. PhD thesis, University of Washington, Department of Computer Science, Seattle, WA, August 1989.

[9] R. E. Cypher and C. G. Plaxton. Techniques for shared key sorting. Unpublished manuscript.

[10] R. E. Cypher and C. G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 193–203, May 1990.

[11] W. Dally and C. Seitz. Deadlock free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C–36(5):547–553, May 1987.

[12] D. S. Greenberg and S. N. Bhatt. Routing multiple paths in hypercubes. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 45–54, July 1990.

[13] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950.

[14] J. Hastad, T. Leighton, and M. Newman. Fast computation using faulty hypercubes. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 251–263, May 1989.

[15] C. Kaklamanis, D. Krizanc, and A. Tsantilas. Tight bounds for oblivous routing in the hypercube. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 31–36, July 1990.

[16] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communications switching technique. *Computer Networks*, 3(4):267–286, 1979.

[17] D. Krizanc, D. Peleg, and E. Upfal. A time-randomness tradeoff for oblivous routing. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 93–102, May 1988.

[18] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. Unpublished manuscript.

[19] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 256–271. IEEE, October 1988.

[20] T. Leighton and G. Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 264–274. IEEE, October 1990.

[21] D. Nassimi and S. Sahni. Parallel algorithms to set up the Benes permutation network. *IEEE Transactions on Computers*, C–31(2):148–154, February 1982.

[22] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2), April 1989.

[23] A. G. Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 185–194. IEEE, October 1987.

[24] E. Upfal. Efficient schemes for parallel communication. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 55–59, August 1982.

[25] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 241–250, May 1989.

[26] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[27] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, May 1981.