Simple Algorithms for Routing on Butterfly Networks with Bounded Queues

(Extended Abstract)

Bruce M. Maggs

NEC Research Institute
Princeton, NJ 08540

Ramesh K. Sitaraman*

Department of Computer Science
Princeton University
Princeton, NJ 08544

Abstract

This paper examines several simple algorithms for routing packets on butterfly networks with bounded queues. We show that for any pure queuing protocol, a routing problem in which each of the N inputs sends a packet to a randomly chosen output requires $O(\log N)$ steps, with high probability, provided that the queue size is a sufficiently large, but fixed, constant. We also show that for any deterministic non-predictive queuing protocol, there exists a permutation that requires $\Omega(N/q \log N)$ time to route, where q is the maximum queue size. We present a new algorithm for routing a random problem on a fully-loaded butterfly with bounded-size queues in $O(\log N)$ steps, with high probability. The algorithm is simpler than the previous algorithms of Ranade and Pippenger because it does not use ghost messages, it does not compare the ranks or destinations of packets as they pass through a switch, and it cannot deadlock. Finally, using Valiant's idea of random intermediate destinations, we generalize a result of Koch's by showing that, if each wire can support q messages, then for any permutation, the expected number of messages that succeed in locking down paths from their origins to their destinations in back-to-back butterflies is $\Omega(N/(\log N)^{1/q})$. The analysis also applies to store-and-forward algorithms that drop packets if they attempt to enter full queues.

1 Introduction

Many commercial and experimental parallel computers, including the NYU Ultracomputer [7], the IBM RP3 [16], the BBN Butterfly [5], and NEC's Cenju [15], use butterfly networks to route packets between processors. Although many routing algorithms with provably good performance have been devised for butterfly networks [2, 14, 17, 20, 21, 26, 27, 28, 29], simpler algorithms are often used in practice. Typically, packets are sent along shortest paths through the network, and simple queuing protocols such as first-in first-out (FIFO) are used to determine which packets to transmit at each step. In addition, the queues at the switches can usually hold only a small number of packets. The performance of these simple algorithms has proven surprisingly difficult to analyze. For example, the only previously known upper bound on the time required for each input of an N-input butterfly network with constant-size FIFO queues to route a packet to a random destination was $O(N)$. This paper shows that the expected time is actually $\Theta(\log N)$. It also analyzes the performance of several other simple algorithms for routing on butterflies with bounded queues.

1.1 Butterfly networks

An example of an N-input butterfly ($N = 8$) with depth $\log N = 3$ is shown in Figure 1. (All logarithms in this paper are to base 2.) The edges of the butterfly are directed from the node in the smaller numbered level to the node in the larger numbered level. The nodes in this directed graph represent switches, and the edges represent wires. We use the terms node and switch interchangeably in the rest of the paper. Each node in a butterfly has a label $\langle l, c_0 \cdots c_{\log N-1} \rangle$, where the *level*, l, ranges from 0 to $\log N$, and the *column*, $c_0 \cdots c_{\log N-1}$, is a $\log N$-bit binary string. The switches on level 0 are called *inputs*, and those on level $\log N$ are called *outputs*. For $l < \log N$, node $\langle l, c_0 \cdots c_l \cdots c_{\log N-1} \rangle$ is connected to node $\langle l+1, c_0 \cdots c_l \cdots c_{\log N-1} \rangle$ by a *straight* edge, and to node $\langle l+1, c_0 \cdots \overline{c_l} \cdots c_{\log N-1} \rangle$ by a *cross* edge. (The notation $\overline{c_l}$ denotes the complement of bit c_l.) At each time step, each switch is permitted to transmit one packet along each of its outgoing edges.

In a butterfly network, packets are typically sent from the inputs on level 0 to the outputs on level $\log N$. If each input sends a single packet, we say that the network is *lightly loaded*. If each input sends $\log N$ packets, we say that it is *fully loaded*. One of the nice properties of the butterfly is that there is a unique path of length $\log N$ between any input and any output, and there is a simple rule for finding that path.

*This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014–88–K–0459.

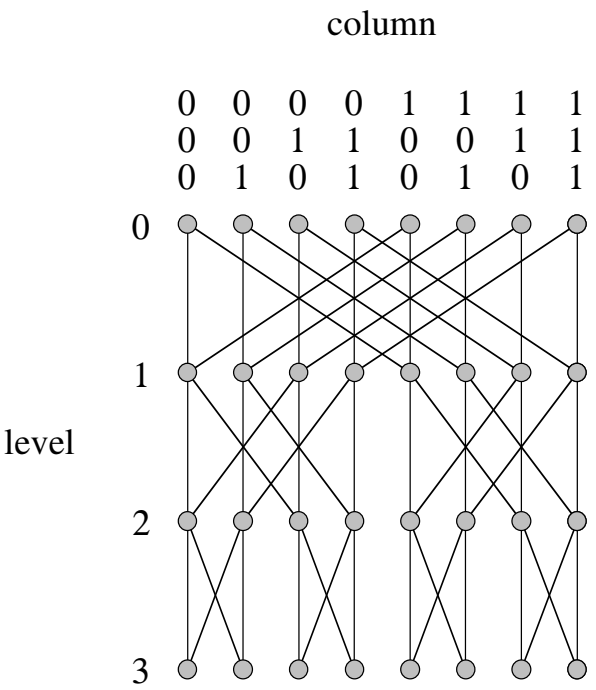

Figure 1: An 8-input butterfly network.

When a packet with origin $\langle 0, a_0 \cdots a_{\log N-1} \rangle$ and destination $\langle \log N, d_0 \cdots d_{\log N-1} \rangle$ reaches level l, it passes through the node labeled $\langle l, d_0 \cdots d_{l-1} a_l \cdots a_{\log N-1} \rangle$. If $d_l = a_l$, then it takes the straight edge to $\langle l + 1, d_0 \cdots d_l a_{l+1} \cdots a_{\log N-1} \rangle$. Otherwise, it takes the cross edge to the same node. This path selection algorithm is called *source oblivious* [6] because, at each node, the next edge taken by a packet depends only on its current location and its destination, and not on its source, or on the locations or paths taken by any of the other packets. All of the routing algorithms discussed in this paper are source oblivious.

1.2 Queuing protocols

This paper studies two broad classes of queuing protocols: pure and non-predictive. Many easily implementable as well as conceptually simple queuing protocols like FIFO and fixed-priority scheduling are pure as well as non-predictive. In a *pure* queuing protocol [10], at each step, each switch with one or more packets in its queue selects a packet, and then sends it to the next level, unless the queue that the packet wishes to enter is already full. A switch is not prohibited from sending more than one packet at each step, provided that they use different edges. In a *non-predictive* queuing protocol [12, 20], at each step, each switch selects one packet from its queue *without examining the destinations of any of the packets in its queue*, and sends the packet to the next level, unless the queue that it wishes to enter is full. If the queue is full, then the switch must select the same packet at the next step. The switch is not permitted to examine the destinations of any other packets until the selected packet has been successfully transmitted. Non-predictive protocols are also pure.

1.3 Previous work

The first important butterfly routing algorithm is due to Batcher [4], who showed that an N-input butterfly network can sort, and hence route, N-packets in $O(\log^2 N)$ steps.

The next breakthrough came more than a decade later when Valiant [27, 29] observed that any permutation routing problem can be transformed into two random problems by routing the packets first to random intermediate destinations, and then on to their true destinations. He also showed that an N-node hypercube (or N-input butterfly) can route N packets to random destinations (or from random origins) in $O(\log N)$ time using queues of size $O(\log N)$, with high probability. As a consequence, the hypercube or butterfly can route any permutation in $O(\log N)$ time, with high probability.

Valiant's result was improved in a succession of papers by Aleliunas [2], Upfal [26], Pippenger [17], and Ranade [13, 21]. All of these papers use Valiant's idea of first routing to random intermediate destinations. Aleliunas and Upfal increased the number of packets that can be routed in $O(\log N)$ time. They developed the notion of a *delay path* and showed how to route N packets on an N-node shuffle-exchange graph and $N \log N$ packets on an N-input butterfly network, respectively, in $O(\log N)$ steps, using queues of size $O(\log N)$. Pippenger devised an ingenious algorithm for routing with bounded size queues. He showed how to route $N \log N$ packets on a variant of the butterfly in $O(\log N)$ steps with queues of size $O(1)$. Finally, Ranade developed a simpler algorithm for routing with bounded queues that could also efficiently combine multiple packets with the same destination. As a consequence of Ranade's algorithm, it is possible to simulate one step of an $N \log N$-processor CRCW PRAM on an N-input butterfly in $O(\log N)$ steps. Neither Pippenger's algorithm nor Ranade's algorithm are pure.

Recently, Stamoulis and Tsitsiklis [22] have analyzed the average delay and queue size in hypercubes and butterflies with unbounded queues in which packets with random destinations are generated according to a Poisson process. They show that if the load factor on the network is less than one, then the network is stable in the steady state, the average delay is $O(\log N)$, and the average queue size is $O(1)$.

Although the performance of pure algorithms in butterflies with bounded queues has proven difficult to analyze, attempts have been made to approximately model [24] or empirically determine [25] their performance.

1.4 Our results

In Section 2 we show that for any pure queuing protocol, a routing problem in which each input in an N-input butterfly sends a single packet to a randomly chosen output requires $O(\log N)$ steps, with high probability, provided that the queue size is a sufficiently-large fixed constant. Previously, only the trivial upper bound of $O(N)$ was known. An intriguing problem left open in this section is a bound on the number steps taken by a pure queuing protocol when the butterfly is fully-loaded.

In Section 3 we show that for any deterministic non-predictive queuing protocol, there exists a one-to-one routing problem (permutation) that requires $\Omega(N/q \log N)$ time to route, where q is the maximum queue size. Previously, no lower bound greater than $\Omega(\sqrt{N})$ was known. The $\Omega(\sqrt{N})$ bound is based on the congestion and is independent of the way the packets are scheduled. This section shows that greater delays can occur due to the way packets interact in the network and the way they are scheduled.

Section 4 presents a simple, but non-pure, algorithm for routing a random problem on a fully-loaded butterfly with bounded-size queues in $O(\log N)$ steps, with high probability. The algorithm is simpler than the previous algorithms of Ranade and Pippenger because it does not use ghost messages, it does not compare the ranks or destinations of packets as they pass through a switch, and it cannot deadlock.

Finally, in Section 5 we analyze routing algorithms that drop packets when there is contention. Examples of machines that drop packets are NEC's ATOM switch [23] and the BBN Butterfly [5]. The BBN Butterfly algorithm has been studied by Kruskal and Snir [11] and Koch [9]. Koch showed that for a random permutation the number of packets that succeed in locking down paths from their origins to their destinations is $\Theta(N/\log^{\frac{1}{q}} N)$, where q is the maximum number of packets that any wire can support. By routing the packets to randomly (but not independently) chosen intermediate destinations, we show that for *any fixed permutation* the expected number of packets reaching their destinations is $\Omega\left(N/\log^{\frac{1}{q}} N\right)$.

2 Pure queuing protocols

In this section, we will study the performance of pure queuing protocols. In Section 2.1, we analyze the average case behavior of any routing algorithm with a pure queuing protocol. We show that if every input sends a packet to a randomly chosen output, then the time required for all of the packets to reach their destinations is $O(\log N)$. In Section 2.2, we show how any specific permutation routing problem on the butterfly can be routed in $O(\log N)$ steps using Valiant's idea of splitting any routing problem into two random routing problems.

2.1 Average case behavior

We first define a few terms. A *delay tree* is a rooted tree that is a subgraph of the butterfly. Its root is a level 0 node and the tree contains a (directed) path, which we call the *spine*, from the root to a node in level $\log N$ of the butterfly. The tree "grows out" and there is a unique directed path in the tree from the root to each node in the tree. A *full node* is defined to be a node through which the paths of at least q packets pass, where q is the maximum size of the queue in each node. Note that in the course of the routing, a full node may never have a full queue since the packets could arrive at different times. However a non-full node can never have a full queue. A *full delay tree* is a delay tree for which every node of the tree that is not on the spine is a full node. The number of *packets on a delay tree* is defined to be the sum over all nodes of the tree of the total number of packets passing through that node. Note that this is different from the number of *distinct packets on a delay tree*. In the former, if a particular packet hits (i.e. passes through) many nodes of a tree it is counted many times in the sum. The significance of the above definitions becomes clear in Theorem 2.1 below.

Theorem 2.1 *The maximum delay of any packet is less than or equal to the maximum of the number of packets on a full delay tree.*

Proof: We will bound the delay of a packet by the number of packets on its full delay tree. Let the path of some packet, from its source to destination, be denoted by P. Now consider the maximal full delay tree of this packet, with the path P as its spine and the source of the packet as its root. Since the tree is maximal, every non-tree node that is a neighbor of a tree node is not a full node. We will now show that at each time step until it reaches its destination, some packet p in this maximal full delay tree moves. At every time step there are 3 cases.

a. The packet p moves.

b. Some other packet in the same node moves.

c. No packet in the same node as p moves.

The first two cases are simple. Since the queuing protocol is pure, case **c** necessarily means that the packet selected by the node to be sent at this time step could not move because the queue in the node, say n, it wanted to enter was full. Note that node n belongs to the maximal full delay tree since it has at least q packets passing through it. Now if some packet in node n moved at that time step we are done. If not we look at the packet selected by node n and repeat the argument again. Note that case **c** cannot apply at the leaves of tree since it does not have any neighbors with full queues. So we

must encounter either case **a** or **b** before we leave the tree.
Therefore the delay of a packet is at most the number of packets on its maximal full delay tree. □

We will use the following property of the butterfly network in the proofs in this section.

Observation 2.2 *A packet can enter the tree at exactly one point and once it leaves the tree can never return to it.*

We also state without proof a Chernoff type bound [3] and [18, p. 56] and a result due to Hoeffding [8].

Lemma 2.3 (Hoeffding) *Let X be the number of successes in r independent Bernoulli trials where the success probability of the i^{th} trial is p_i. Let S be the number of successes in r independent Bernoulli trials where each trial has probability of success $p = \frac{1}{r}\sum_{1\leq i\leq r} p_i$. Then $E(X) = E(S) = rp$, and*

$$\Pr[X \geq \alpha E(X)] \quad \leq \quad \Pr[S \geq \alpha E(S)] \qquad (1)$$

for $\alpha E(S) \geq E(S) + 1$.

Lemma 2.4 (Chernoff) *Let S be the number of successes in r independent Bernoulli trials where each trial has probability p. The $E(S) = rp$, and*

$$\Pr[S \geq \alpha E(S)] \quad \leq \quad 2^{-\alpha E(S)} \qquad (2)$$

for $\alpha > 2e$.

Theorem 2.5 *Let constant q be the maximum queue size. Then the maximum delay of any packet is at most $\gamma \log N$ with probability at least $1 - \frac{1}{N}$, for some sufficiently large but constant γ and q.*

Proof: We will show that if there is a packet with large delay, then there must be a delay tree with a large number of packets on it, which in turn we will show to be an unlikely event. Assume that some packet has a delay $\gamma \log N$ or more. Let D denote the number of hits on the delay tree. Consider the maximal full delay tree of this packet. By Theorem 2.1, we know that $D \geq \gamma \log N$. Also since every non-spine node of this delay tree is necessarily a full node, the maximum number of nodes of this maximal full delay tree is at most $\frac{D}{q} + \log N$. Let n be a node on any level l of the butterfly. The average number of packets passing through n is 1, because there are 2^l possible packets that can pass through this node and each of these packets has a probability of 2^{-l} of passing through it. Therefore expected number of packets on this delay tree of is at most $\frac{D}{q} + \log N$. The gist of the proof is to show that the number of packets on this tree is clustered around its mean and therefore unlikely to have D packets on it, for sufficiently large constants q and γ.

Let us distinguish the hits on a delay tree into two kinds : b-hits (for big hits) which are hits made by packets making at least c hits on the tree and s-hits (for small hits) which are hits made by packets making less than c hits on the tree, where c is some constant. It must be the case that either the total b-hits on some delay tree is greater than or equal to $\frac{D}{2}$ (call this event E_b) or the total s-hits on some delay tree is greater than or equal to $\frac{D}{2}$. The latter possibility also implies that there are at least $\frac{D}{2(c-1)}$ distinct packets hitting some delay tree (call this event E_s), since each packet making s-hits can make at most $c-1$ hits on the tree. Thus, the probability that some packet has delay d is at most $\Pr(E_b) + \Pr(E_s)$. The intuitive reason as to why b-hits are unlikely is as follows. If you imagine packets running backwards in time from destination to source, once a packet enters the tree, it can remain in the tree at the next step only if it takes the unique edge to its ancestor in the tree. So, at every step, it has approximately a probability of $\frac{1}{2}$ of making another hit. This exponentially decreasing probability for making more and more hits gives us the bound. Thus, this bound uses the fact that it is a tree in a crucial way, unlike the bound for E_s which is valid for any set of $\frac{D}{q} + \log N$ nodes.

Bounding the big hits: Let us suppose that event E_b occurs, i.e., there exists a delay tree of size at most $\frac{D}{q} + \log N$ with a total of at least $\frac{D}{2}$ b-hits. To bound the probability of this event we will enumerate all the possible ways it can happen.

The maximum value that D can take is $N \log N$, since each packet can contribute at most $\log N$ hits and there is a total of N packets. Therefore, the number of ways of choosing D is at most $N \log N$. The number of ways of choosing the root for the delay tree is N. Now one can represent a binary tree of size at most $\frac{D}{q} + \log N$ by indicating the number of children (no children, left son only, right son only, both sons) in breadth first search order. Thus the total number of ways of choosing the delay tree is at most $N 4^{\frac{D}{q} + \log N}$. The number of different packets causing these b-hits are at most $\frac{D}{c}$, since each packet needs to cause at least c hits and there are a total of at most D hits on the tree. Let us assume that there is some arbitrary fixed ordering of the nodes in the tree, e.g., the breadth first search ordering of the tree. We will now pick a nondecreasing sequence of $\frac{D}{c}$ nodes in the tree, $n_1, n_2, \cdots, n_{\frac{D}{c}}$. Note that each node of the tree can occur more than once in this sequence. The sequence represents the last switches on the tree through which the packets which hit the tree passed. The number of ways of choosing this sequence at most

$$\left(\begin{array}{c} \frac{D}{q} + \log N + \frac{D}{c} \\ \frac{D}{q} + \log N \end{array} \right)$$

Let node n_i of the sequence be at level l_i of the butterfly. For every n_i, we now associate a non-negative integer h_i denoting the number of hits made by a packet p_i before leaving node n_i. The number of ways of distributing at

most D hits over $\frac{D}{c}$ elements of the sequence is at most $\binom{D+\frac{D}{c}}{\frac{D}{c}}$. We can ignore n_i with $h_i = 0$ in this. Since the packet must have made exactly h_i hits before leaving the tree at n_i, the number of choices for p_i is $2^{l_i - h_i}$. Here we have used Observation 2.2. The total number of ways of choosing packets for all elements in the sequence is at most $\prod_i 2^{l_i - h_i}$. (We are overcounting a little since packets have to be distinct.) Now, we have chosen a particular tree, a sequence of nodes n_i and the associated packets p_i. The probability that all the packets p_i pass through their corresponding nodes n_i is simply the product of the probabilities that each individual packet p_i passes through node n_i which equals $\prod_i 2^{-l_i}$. (We can multiply probabilities because each packet chooses its path independently.) Putting it all together, we have

$$\begin{aligned}
\Pr(E_b) &\leq N \log N \cdot N 4^{\frac{D}{q}+\log N} \cdot \binom{\frac{D}{q}+\log N+\frac{D}{c}}{\frac{D}{q}+\log N} \\
&\quad \cdot \binom{D+\frac{D}{c}}{\frac{D}{c}} \cdot \prod_i 2^{l_i - h_i} \cdot \prod_i 2^{-l_i} \\
&\leq N^5 2^{2\frac{D}{q}} \cdot \left(\frac{(\frac{D}{q}+\log N+\frac{D}{c})e}{\frac{D}{q}+\log N} \right)^{\frac{D}{q}+\log N} \\
&\quad \cdot \left(\frac{(D+\frac{D}{c})e}{\frac{D}{c}} \right)^{\frac{D}{c}} \cdot 2^{-\sum_i h_i} \\
&\leq 2^{5\frac{D}{\gamma}} \cdot 2^{2\frac{D}{q}} \cdot \left(\left(1+\frac{q}{c}\right) e \right)^{\frac{D}{q}+\frac{D}{\gamma}} \\
&\quad \cdot 2^{\log((c+1)e)\frac{D}{c}} \cdot 2^{-\frac{D}{2}} \qquad (3)
\end{aligned}$$

since $D \geq \gamma \log N$ and $\sum_i h_i \geq \frac{D}{2}$. Note that the multiple of D in the exponent of the first four factors decreases with an increase in the values of c, q and γ. So for some suitably large values for the constants c, q and γ the expression in equation 3 is at most 2^{-kD} for constant $k > 0$. Now increase γ further if necessary so that we can use the fact that $D \geq \gamma \log N$, and bound the value of this expression (and hence $\Pr(E_b)$) to be at most $\frac{1}{2N}$.

Bounding the small hits: Let us suppose event E_s occurs, i.e. there is a tree of size at most $\frac{D}{q}+\log N$ with at least $\frac{D}{2(c-1)}$ different packets hitting the tree. Let X denote the total number of distinct packets hitting this tree. X is a sum of N boolean random variables, $X_i, 1 \leq i \leq N$. Each X_i is 1 if the packet originating at input i hits the tree and 0 otherwise. The expected number of distinct packets on the tree is at most the expected number of packets on the tree. Therefore, $E(X) \leq \frac{D}{q}+\log N$. Using Lemmas 2.3 and 2.4, we have

$$\begin{aligned}
\Pr(E_s) &= \Pr(X \geq \frac{D}{2(c-1)}) \\
&\leq 2^{-\frac{D}{2(c-1)}} \qquad (4)
\end{aligned}$$

as long as $\alpha = \frac{\frac{D}{2(c-1)}}{E(X)} > 2e$. Using the fact that $D \geq \gamma \log N$, we have

$$\alpha \geq \frac{\frac{D}{2(c-1)}}{\frac{D}{q}+\log N} \geq \frac{q\gamma}{2(c-1)(q+\gamma)} \qquad (5)$$

For any value of c, by choosing large enough values for q and γ we can satisfy the condition that $\alpha > 2e$. The number of ways of choosing a value for D is at most $N \log N$. The number of ways of choosing a tree is at most $N 4^{\frac{D}{q}+\log N}$. Therefore,

$$\Pr(E_s) \leq N \log N \cdot N 4^{\frac{D}{q}+\log N} \cdot 2^{-\frac{D}{2(c-1)}}$$

For a sufficiently large value of q and γ, this expression is at most 2^{-jD} for constant $j > 0$. Now increase γ further if necessary so that we can use the fact that $D \geq \gamma \log N$, and bound the value of this expression (and hence $\Pr(E_s)$) to be at most $\frac{1}{2N}$.

Note that the values of constants c, q and γ were required to be "sufficiently large" in various portions of the proof. They must be chosen to be the maximum of that required for the various steps in the proof. It now follows that the probability that a packet has delay d greater than $\gamma \log N$ is at most $\frac{1}{2N} + \frac{1}{2N}$ which equals $\frac{1}{N}$. □

2.2 Routing a fixed permutation

The results of Section 2.1 deal with the routing delay of an average routing problem. What can we say about routing a fixed permutation? We can show that we can route any fixed permutation such that the routing completes in $O(\log N)$ steps with a high probability using Valiant idea of routing in two phases. In phase A, each packet is routed from its source in level 0 to a random intermediate destination in level $\log N$. Generally the nodes in level $\log N$ are identified with the corresponding nodes in level 0. The packets are queued up at the end of phase A and in phase B each packet is routed to their actual destination.

Theorem 2.6 *Any fixed permutation can be routed using Valiant's paradigm such that the delay is $O(\log N)$ with a probability $\geq 1 - \frac{2}{N}$.*

Proof: Phase A is precisely the same problem as that studied in Section 2.1. In phase B, each packet is routed from its intermediate destination to its final destination. For convenience, we will denote the level of its final destination as 0 and that of the intermediate destination as level $\log N$. This phase is a little different from the one we studied in that the starting points are random while the destinations are fixed. But the same proofs for the delay will work with small modifications. It is perhaps best to imagine the packets running backwards from level 0 (final destinations) to random nodes

in level $\log N$ (intermediate destinations). In the proof for bounding the b-hits, the sequence n_i will now represent switches through which packets that hit the tree, *entered* the tree (running backwards in time). The number of ways of associating a packet with n_i in level l_i is 2^{l_i}. The probability that the packet makes h_i hits is now $2^{-(l_i+h_i)}$, since it must leave the tree at the unique ancestor of n_i in level $l_i - h_i + 1$. The rest of the calculation is the same as before. The proof for bounding the s-hits is identical. □

3 Difficult permutations

In this section, we prove that for any deterministic non-predictive queuing protocol, there exists a permutation that requires $\Omega(N/q \log N)$ time to route on a butterfly network. Previously, the best lower bound for routing on a butterfly with queues of any size was $\Omega(\sqrt{N})$. The $\Omega(\sqrt{N})$ bound is proved by observing that certain permutations, such as the bit-reversal permutation, force $\Omega(\sqrt{N})$ packets to pass through a single switch [12, Section 3.4.2]. (It is also not very difficult to prove that if the queue size is not bounded, then $O(\sqrt{N})$ is an upper bound on the time to route any permutation using any pure protocol.) Because the $\Omega(\sqrt{N})$ bound is based on congestion only, it applies to any queuing protocol. The results in this section indicate that the manner in which packets are scheduled can potentially cause much greater delays. The proof involves a careful examination of the interaction of the packets as they route through the network.

To simplify the presentation in this section, we will assume that each switch has a single queue, and that, at each step, its two neighbors at the previous level may each send a packet into the queue provided that, at the beginning of the step, the queue held at most q packets. We call q the *queue threshold* of the switch. Since a queue can receive 2 packets when it already has q, it may contain as many as $q + 2$ packets, but no more.

Theorem 3.1 *For any deterministic non-predictive queuing protocol, there exists a permutation π that requires $\Omega(N/q \log N)$ steps to route on a butterfly with queue threshold q.*

Proof: The proof is by induction. We will assume that there are two edges leading into each butterfly input, and we begin by computing the time, $t_d(r)$, required for a depth d butterfly to accept $r/2$ packets on each of the 2^{d+1} edges into its inputs. (For simplicity, we assume without loss of generality that r and q are even.) We will assume that at time step 1 and at each time step thereafter, 1 packet is available for transmission along each of these edges until $r/2$ packets have crossed the edge. Furthermore, we will assume that each output switch can transmit one packet at each step.

We begin by examining a 1-input butterfly, which consists of a single switch, s. Suppose that at the beginning of time step 1, the queue at switch s is empty. We would like to know how long takes for s to receive $r/2$ packets from each of its incoming edges, where $r > q$. On time steps 1 through q, s receives one packet along each of its two incoming edges. During steps 2 through q, s transmits one packet at each step. Thus, after q steps, $2q$ packets have been received, $q - 1$ have been transmitted, and the queue contains $q+1$ packets. Since the queue is full, s does not receive any packets on step $q+1$, but it does transmit one. Thereafter, s receives two packets on every other step, and transmits one packet on every step, until a total of r packets have been received, which occurs on step $q + (r - 2q) = r - q$. Thus, $t_0(r) = r - q$.

Next, let us compute the time required for each input of a depth-d butterfly to receive $r/2$ packets along each of its incoming edges. In order for an input to receive r packets, it must transmit at least $r-(q+2)$ packets. Using the assumption that the queuing protocol is nonpredictive, we will choose the paths of these $r-(q+2)$ packets so as to maximize the delay. Since a switch cannot look at a packet's destination until it has been selected for transmission, we can wait until a packet has been selected, and then decide if it should take a cross edge or a straight edge to the next level. The first $(r-(q+2))/2$ packets selected by each input switch will be sent to the switches labeled $\langle 1, 0c_1 \cdots c_{d-1}\rangle$. These switches are the inputs of a depth-$(d-1)$ sub-butterfly. The second $(r-(q+2))/2$ packets will be sent to the depth-$(d-1)$ sub-butterfly whose inputs are labeled $\langle 1, 1c_1 \cdots c_{d-1}\rangle$.

The inputs of the first sub-butterfly start accepting packets at step 2. By induction, the time required for each input to receive $r-(q+2)$ packets is $t_{d-1}(r-(q+2))$. Thus, the first sub-butterfly receives packets during steps 2 through $t_{d-1}(r-(q+2))+1$. In the meantime, no packets are sent to the inputs of the second sub-butterfly. The first packets arrive there on step $t_{d-1}(r-(q+2))+2$, and continue to arrive until step $2t_{d-1}(r-(q+2))+1$, at which point each input has received $r-(q+2)$ packets. Thus, $t_d(r) = 2t_{d-1}(r-(q+2))+1$. Solving this recurrence yields

$$\begin{aligned} t_d(r) &\geq 2^d t_0(r-(q+2)d) \\ &\geq 2^d(r-(q+2)(d+1)). \end{aligned}$$

The lower bound on $t_d(r)$ that we have just derived requires $r > (q+2)(d+1)$ packets to pass through each butterfly input. In a permutation routing problem, however, only one packet originates at each input. In order to use the bound, we will force r packets through each input of an N/r^2-input sub-butterfly that spans levels $\log r$ through $\log N - \log r$. We call this sub-butterfly the *busy sub-butterfly*. It has depth $d = \log N - 2\log r$. Each input of this sub-butterfly is the root of a depth-$\log r$ complete binary tree whose leaves are butterfly inputs on level 0. Well call these trees the *input trees*. Each output is the root of a $\log r$-depth complete binary tree whose leaves lie on level $\log N$. We call these trees the

output trees. All of these trees are completely disjoint. The r packets that originate at the leaves of an input tree will all be sent through the root of that tree. Each output of the busy sub-butterfly receives exactly r packets. These packets are distributed among the r leaves of the corresponding output tree so that they each receive exactly one packet. Note that between levels $\log r$ and $\log N - \log r$, the only switches and edges used for routing are those in the busy sub-butterfly.

All that remains is to choose appropriate values of r and d. From the construction of the busy sub-butterfly, we know that $d = \log N - 2\log r$. In order for our lower bound on t_d to be greater than zero, we need $r > (q+2)\,(d+1)$. Choosing $r = 2(q+2)(\log N+1)$ yields $t_d(r) \geq 2^d(q+2)(\log N+1) = (N/r^2)(q+2)(\log N+1) = N/(4(q+2)(\log N+1))$. Thus the delay is $\Omega(N/q \log N)$. □

Note that the maximum number of packets passing through any node (the congestion) for the worst-case permutation constructed in this section is only $O(\log N)$. This implies that there are other more complex routing algorithms like that of Ranade [21] which can route this permutation in $O(\log N)$ steps!

4 A simple routing algorithm

In this section we present a simple, but non-pure, algorithm for routing on butterfly networks. With high probability, the algorithm requires $O(k+\log N)$ time to route packets with random destinations, where k is the number of packets that originate at each input. The algorithm is simpler than the algorithms of Pippenger [17] and Ranade [21] because it does not use ghost messages, it does not compare the ranks or destinations of packets as they pass through a switch, and it cannot deadlock. Unlike the algorithm of Ranade, however, it does not combine packets with the same destination.

The routing algorithm begins by breaking the packets into *waves*. Each input contributes one packet to each wave. The waves of packets are separated by waves of *tokens*. Unlike the ghost messages in Ranade's algorithm, a token carries no information other than its type, which requires $O(1)$ bits to represent.[1] Initially, there are k packets at each input and a token is placed between each pair of successive packets, and after the last packet. For $0 \leq i \leq k-1$, the ith packet at each input is assigned to wave $2i$, and the ith token is assigned to wave $2i+1$. Thus, the packets belong to the even waves, and the tokens to the odd waves. Throughout the course of the routing, the algorithm maintains the following important invariant. For $i < j$, no packet or token in the jth wave leaves a switch before any packet or token in the ith wave. Furthermore, packets within the same wave pass through a switch in the increasing order of their column numbers of origin. (A column $c_0 \cdots c_{\log N-1}$ is viewed as a binary number where c_0 is the lower order bit.)

A switch labeled $\langle l, c_0 \cdots c_{\log N-1}\rangle$ has two edges into it, one from the switch labeled $\langle l-1, c_0 \cdots c_{l-2}\,0\,c_l \cdots c_{\log N-1}\rangle$, and the other from the switch labeled $\langle l-1, c_0 \cdots c_{l-2}\,1\,c_l \cdots c_{\log N-1}\rangle$. We call the first edge the *0-edge*, and the other the *1-edge*. At the end of each of these edges is a first-in first-out queue that can hold q packets. We call these queues the 0-queue and the 1-queue, respectively.

The behavior of each switch is governed by a simple set of rules. By "forward" a packet or token we mean send it to the appropriate queue in the next level. If that queue is full, the switch tries again in consecutive time steps until it succeeds. A switch can either be in 0-mode or in 1-mode and is initialized to be in 0-mode. In 0-mode, a switch forwards packets in the 0-queue in FIFO fashion, until a token is at the head of the 0-queue. It then changes to the 1-mode. In the 1-mode, a switch forwards packets in the 1-queue in FIFO fashion, until a token is at the head of the 1-queue as well. Now the switch waits until both the queues at its outgoing edges can receive a token and then simultaneously sends one token to each of them. After doing this, the switch changes back to the 0-mode.

Note that at each step a switch is required to perform only $O(1)$ bit operations in order to determine which packet, if any, to send out. In the algorithms of Pippenger and Ranade, the switches must perform more complicated operations, such as comparing the destinations of two packets as they pass through a switch. In the succeeding sections, we show that our algorithm requires $O(k+\log N)$ steps, which is asymptotically optimal.

4.1 Delay sequences

The proof that the algorithm requires $O(k+\log N)$ time uses a delay sequence argument similar to those in [1, 13, 21]. A (w,f)-delay sequence consists of four components: a path P from an output to an input; a sequence $s_1, \ldots, s_w$ of w, not necessarily distinct, switches which appear in order on the path; a sequence $h_1, \ldots, h_w$ of w distinct packets and tokens; and a non-increasing sequence of wave numbers $r_1, \ldots, r_w$. The path P may trace any edge of the network in either direction. When the path traces an edge from some level l to level $l+1$, we call the edge a *forward* edge. The number of forward edges in the path is denoted by f. The length, L, of P is equal to the distance from an output to an input ($\log N$) plus two times the number of forward edges on P, $L = \log N + 2f$. We say that a delay sequence *occurs*, if, for $1 \leq i \leq w$, packet or token h_i belongs to wave r_i, and passes through switch

[1] Tokens are used in a similar fashion in a bit-serial algorithm for routing on the hypercube in [1]. It turns out, however, that tokens are not really needed in that algorithm. Ranade's proof of the equivalence of different queuing disciplines [20] implies that a first-in first-out queuing protocol will suffice.

s_i. The following lemma shows that if some packet is delayed, then a delay sequence must have occurred.

Lemma 4.1 *If some packet arrives at its destination at step* $\log N + d$ *or later, then a* $(d + (q-2)f, f)$*-delay sequence must have occurred, for some* $f \geq 0$*. Furthermore, no two tokens in the sequence belong to the same wave.*

Proof: Before we begin the proof, we need some definitions. Let the *lag* of a switch s at time t on level l be $t - l$. Also, let the *rank* of a packet h be a 2-tuple consisting of h's wave number and the column number of the input in which it originated. Ranks are compared by first comparing wave numbers, and then, if there is a tie, comparing column numbers. A column $c_0 \ldots c_{\log N - 1}$ is viewed as a binary number where c_0 is the low order bit. Note that each packet has a distinct rank. Every token belonging to the same wave has the same rank. This rank is strictly less than all the packets in the wave above it but strictly greater than the packets in the wave below it. Note that ranks are used only as a tool for the analysis and not by the algorithm itself.

The algorithm maintains several important invariants. As mentioned before, the packets and tokens leave each switch in order of non-decreasing wave number. Furthermore, each edge transmits exactly one token from each odd wave. Finally, within an even wave, the packets that arrive at a switch via its 0-edge have smaller ranks than the packets that arrive via its 1-edge. As a consequence, each switch sends out packets and tokens in order of strictly increasing rank.

The delay sequence begins with the last packet to arrive at its destination. Suppose that some packet h_1 arrives at its destination, s_1, at step τ_1, where $\tau_1 \geq \log N + d$. Then s_1 has lag at least d at step τ_1. We will construct the delay sequence by spending lag points. We begin the sequence with h_1, s_1, and r_1, where r_1 is the wave number of h_1. Next, we follow h_1 back in time until the step at which it was last delayed.

In general, suppose that we have followed some packet or token h_i back in time from some switch s_i at time step τ_i until it was last delayed, at some switch s'_{i+1} at time step τ_{i+1}. Because h_i is delayed at s'_{i+1} at step τ_{i+1}, the lag at s'_{i+1} at step τ_{i+1} is one less than the lag of s_i at step τ_i. There are three possible reasons for the delay of h_i at switch s'_{i+1}.

First, if s_{i+1} selects another packet or token, h_{i+1}, to send instead of h_i, then h_{i+1} must have a strictly smaller rank than h_i. In this case, h_{i+1}, $s_{i+1} = s'_{i+1}$, and r_{i+1} are inserted into the sequence, where r_{i+1} is the wave number of h_{i+1}. We then follow h_{i+1} back in time until it was last delayed. We have spent one lag point, and inserted one packet or token into the sequence.

Second, if s'_{i+1} doesn't send h_i because the queue at the end of one of its outgoing edges is full, then we extend the path, P, forward along that edge to the switch at its head, s''_{i+1}. The lag of switch s''_{i+1} at time τ_{i+1} is two less than the lag of s_i at step τ_i. However, the queue must contain a total of q packets and tokens, all of which have smaller rank than h_i. We insert these packets and tokens into the sequence. We then follow the packet or token at the head of the queue back in time until it was last delayed.

If neither of these cases is true, it must be the case that in switch s'_{i+1} at time τ_{i+1} either of the following occurs.

(a) h_i is a packet and it is at the head of the 1-queue and the 0-queue is empty, or

(b) h_i is a token and it is at the head of one of the queues and the other queue is empty.

In either case, we go back to the switch at the tail of the empty queue at the previous time step. Note that we do not loose any lag by this process. We continue to do this as long as we can find an empty queue at the current switch. Suppose we do it m times and we are at a switch s''_{i+1} at time $\tau_{i+1} - m$. Switch s''_{i+1} has packets or tokens at the head of both its queues but did not send anything through one of its edges at time $\tau_{i+1} - m$. If one of the heads of its queues is a packet, we add it and switch s''_{i+1} to the delay sequence and continue to follow this packet back in time. Note that in case (a), this packet belongs to the same wave as h_i but has rank strictly less than it since the first edge we followed up is a 0-edge. In case (b), the packet belongs to a wave earlier than that of token h_i and hence has a strictly smaller rank. In either case, we have added a packet of strictly smaller rank for the cost of one lag point. Now suppose that both the heads of queues are tokens. The only reason the tokens were not sent at time $\tau_{i+1} - m$ is that one of the outgoing edges of s''_{i+1} has a full queue. In this case we extend the path P forward to the switch at the head of the queue, and insert all of the packets in that queue into the delay sequence and follow the packet or token at the head of the queue back in time. Now we have added q packets and tokens for the cost of two lag points.

For each lag point spent, at least one new packet or token is inserted into the delay sequence. Furthermore, for each forward edge on the path P, an additional $q-2$ packets and tokens are inserted. Let f be the number of forward edges on P. Since we had d lag points to spend, we must insert at least $d + (q-2)f$ packets and tokens. Since we are inserting packets or tokens in strictly decreasing order of rank, at most k of these can be tokens. The length of P is $\log N + 2f$. □

4.2 Bunched delay sequences

We have now established that if some packet is delayed, then a delay sequence occurs. To simplify the rest of the argument, we will restrict our attention to delay sequences in which the packets can be partitioned into

bunches of size b such that all of the packets in each bunch pass through the same switch on the sequence and have the same wave number. We call such a delay sequence a *bunched* delay sequence. The following lemma shows that if a delay sequence occurs, then a bunched subsequence also occurs.

Lemma 4.2 *If a $(d+(q-2)f, f)$ delay sequence occurs, then a (bg, f) bunched delay sequence occurs, where*

$$g = \left\lceil \frac{d + (q-2b)f - bk - (b-1)\log N}{b} \right\rceil .$$

Proof: Suppose that a $(d+(q-2)f, f)$ delay sequence occurs. We will describe an algorithm for finding a bunched subsequence.

Starting at the first switch on the sequence, s_1, form a bunch of size b of packets with wave number $2(k-1)$. If successful, then form another bunch of packets with wave number $2(k-1)$. Otherwise, if there are fewer than b remaining packets with wave number $2(k-1)$, then there are two cases to consider. First, if there are other packets on the sequence that pass through s_1, then discard the remaining packets with wave number $2(k-1)$, and begin forming bunches out of packets with the next smaller even wave number. Since the wave number can decrease at most k times, this case can happen only k times. Each time, we may discard as many as $b-1$ packets from the original delay sequence. Second, if no other packets on the sequence pass through s_1, then move on to the second switch, s_2. This case can happen at most $\log N + 2f$ times, since the path has length $L = \log N + 2f$. As in the first case, we may discard $b-1$ packets from the original sequence.

Since the original sequence contains at least $d+(q-2)f-k$ packets, and we discard a total of at most $k(b-1)+(\log N+2f)(b-1)$ packets, at least $d+(q-2b)f-bk-(b-1)\log N$ packets are placed in bunches. Thus, there are at least $g = \left\lceil \frac{d+(q-2b)f-bk-(b-1)\log N}{b} \right\rceil$ bunches. □

4.3 The counting argument

We are now in a position to prove that, with high probability, every packet reaches its destination within $O(k + \log N)$ steps.

Theorem 4.3 *For any c_2 and $q > 0$, there exists a constant c_1 such that the probability that any packet is delayed for more than $d = c_1(k+\log N)$ steps is at most $1/N^{c_2}$, where k is number of packets per input of the butterfly.*

Proof: To prove this theorem we will enumerate all possible bunched delay sequences, and show that it is unlikely that any of them occurs.

The number of different bunched delay sequences is at most

$$N \cdot 4^L \cdot \binom{L+g}{g} \cdot \binom{g+k}{g} \cdot \prod_{i=1}^{g} \binom{2^{d_i}}{b}, \qquad (6)$$

where d_i is the level of the switch through which the ith bunch passes. The factors in this product are explained as follows. There are N choices for the output switch at which the path P in the delay sequence originates. At each of the first L switches on the path, there are at most 4 choices for the next switch on the path. There are at most $\binom{L+g}{g}$ ways of choosing the g (not necessarily distinct switches) on the path that the g bunches pass through, and at most $\binom{g+k}{g}$ ways of choosing (not necessarily distinct) wave numbers for the g bunches. Finally, given a switch with level d_i, and wave number w, there are $\binom{2^{d_i}}{b}$ ways of choosing b packets with wave number w that can pass through the switch.

Whether or not a particular delay sequence occurs depends entirely on the random destinations chosen by the packets in the delay sequence. It is important to note that every packet on the delay sequence is distinct. Therefore the events regarding any two packets on the delay sequence are mutually independent. Thus the probability that all of the packets pass through their corresponding switches is

$$\prod_{i=1}^{g} \frac{1}{2^{bd_i}}, \qquad (7)$$

since each of the b packets in the ith bunch has probability $1/2^{d_i}$ of passing through any particular switch on level d_i.

We can bound the probability that *any* delay sequence occurs by summing the probabilities of each individual delay sequence occurring, which is equivalent to multiplying (6) by (7). Using the inequality $\binom{x}{y} \le (ex/y)^y$ to bound $\binom{L+g}{g}$ and $\binom{g+k}{g}$, and using $\binom{x}{y} \le x^y/y!$ to bound $\binom{2^{d_i}}{b}$, the product is at most $2^{3\log N+4f} \cdot (e(L+g)/g)^g \cdot (e(g+k)/g)^g \cdot (1/b!)^g$, where $g = \left\lceil \frac{d+(q-2b)f-(b-1)k-(b-1)\log N}{b} \right\rceil$. By making q large compared to b (but still constant), and d large compared to $b(k+\log N)$ (but still $c_1(k+\log N)$, where c_1 is a constant), we can make g larger than L, k and $3\log N+4f$. In this case, the product is at most $(8e^2/b!)^g$. By making b large enough, we can make this product less than $1/N^{c_2}$, for any constant c_2. □

5 Algorithms that drop packets

In this section, we consider queuing protocols that resolve contention by dropping packets. Two examples of machines that use this kind of protocol are the BBN Butterfly [5] and the NEC ATOM switch [23].

The ATOM switch routes packets in a store-and-forward manner. At every time step, each switch examines the head of its input queue and forwards a packet to the appropriate output queue. If a queue receiving packets is already full then it discards packets in excess of its maximum queue size. The BBN Butterfly operates in the circuit-switching paradigm. Each packet tries to lock down a path between its source and destination. We will assume that each edge of the butterfly can sustain up to q such paths. This means that some packets may not be able to lock down paths to their destinations. In both of these queueing protocols, a natural question to ask is how many of the packets reach their destinations. The ATOM switch has not been studied in this context before. Koch [9] studied the average case scenario for the BBN Butterfly algorithm and showed that if each packet independently chooses a random destination then the expected number of packets which get through is $\Theta(N/\log^{\frac{1}{q}} N)$. However there are permutations that arise from natural problems in which only $O(\sqrt{N})$ packets get through. To combat this we show how to route any *fixed* permutation in either of the above mentioned queueing protocols such that the expected number of packets that reach their destinations is $\Omega\left(N/\log^{\frac{1}{q}} N\right)$. As an aside, this section also provides an elementary proof of the fact that the expected number of packets that get through for a random routing problem on the butterfly is $\Omega\left(N/\log^{\frac{1}{q}} N\right)$. As mentioned earlier, this was first proved by Koch [9].

The idea for routing any fixed permutation is based on Valiant's idea of routing to random intermediate destinations. Consider two back-to-back butterflies, i.e. two butterflies whose level $\log N$ nodes are identified. The source of the packets is level 0 of one of the butterflies and each packet has a destination in level 0 of the other butterfly. In the first stage, the packets route from level 0 to level $\log N$ of the first butterfly. In the second stage of the routing, the packets route from level $\log N$ to level 0 of the second butterfly. In the first stage we use a scheme for sending packets to random but not independent destinations. Ranade[19] was the first to use this scheme in order to reduce the amount of the randomness needed to send packets to intermediate destinations in a packet switching algorithm. The scheme is as follows. At time step i every level i switch receives two packets, one from each of its incoming butterfly edges. The switch selects a random outgoing edge for one of the packets and routes the other packet through the remaining outgoing edge. Therefore in the first stage no packets are dropped. In the second stage, every packet is routed from this intermediate destination to its actual destination in level 0 of the second butterfly. In this stage, packets are dropped according to the rules of BBN Butterfly routing or that of the ATOM switch. We will assume that each packet picks a random rank from 1 to $r = \log^{\frac{1}{q}} N$. When packets need to be dropped, packets with the least rank are dropped in favor of those with a higher rank. We will now show that the expected number of packets which reach their destination is $\Omega\left(N/\log^{\frac{1}{q}} N\right)$.

Let n be a node at level l of the second butterfly. Consider any k packets whose final destinations are reachable from this node. We bound the probability that all k packets pass through this node.

Lemma 5.1 *The probability that all k packets pass through a node n in level l of the second butterfly is at most $\frac{1}{2^{lk}}$.*

Proof: Let the node in the first butterfly which corresponds to node n be n'. Let the sub-butterfly from level l to n of the first butterfly which contains n' be B. Note that the k packets pass through the given node n if and only if all of these packets pass through some node of sub-butterfly B. Consider the sources of the k packets in level 0 of the first butterfly and the unique shortest paths from each of the sources to sub-butterfly B. If any two of them intersect before reaching the sub-butterfly these two packets cannot both hit sub-butterfly B, since at the node of intersection only one packet can take the path to the sub-butterfly. If no two paths intersect, then the probability of each packet hitting B is independent of the others and equals $\frac{1}{2^l}$. Thus in this case the probability of all them hitting the sub-butterfly is exactly $\frac{1}{2^{lk}}$. Therefore given any k packets the probability of all k of them passing through the given node or equivalently hitting the sub-butterfly is at most $\frac{1}{2^{kl}}$. □

Theorem 5.2 *The expected number of packets to reach their destinations is $\Omega\left(N/\log^{\frac{1}{q}} N\right)$.*

Proof: Consider the path of a particular packet in the second butterfly. We will now evaluate the probability that this packet reaches its destination. Note that with probability $\frac{1}{r}$ this packet will have the highest rank r. In this case, this packet can be dropped only if there is node on its path with at least q packets going through it, all with rank r. We will now show a lower bound on the probability that there exist no such q packets. First let's bound, E_q, the expected number of q-tuples of packets incident on a node at level l of the second butterfly.

$$\begin{aligned} E_q &\leq \binom{2^l}{q} \frac{1}{2^{ql}} \\ &\leq \frac{1}{q!}, \end{aligned} \tag{8}$$

since there are $\binom{2^l}{q}$ ways of choosing q packets that can pass through a node on level l, and by Lemma 5.1, the probability that these packets actually pass through the node is at most $1/2^{ql}$.

The expected total number of q-tuples incident on some node on the path is at most $\log N/q!$. The expected number of such q-tuples with all packets having rank r is at most $\log N/(q!r^q)$ which equals $1/q!$, since $r = \log^{\frac{1}{q}} N$. As $1/q! \leq 1$, the probability that no such q-tuple exists anywhere on the path of the packet is at least $1 - 1/q!$. (A slightly bigger choice for r would make the arguments work for $q = 1$.) This implies that the packet reaches its destination with a probability of at least $(1 - 1/q!)(1/r)$, since the probability that the packet gets rank r is $1/r$. Therefore the expected number of packets to reach their destinations is at least $(1 - 1/q!)(N/r) = \Omega\left(N/\log^{\frac{1}{q}} N\right)$. □

The proof that the expected number of packets to reach their destinations is $\Omega\left(N/\log^{\frac{1}{q}} N\right)$ also holds for a random routing problem in which each packet chooses independently a random destination. Lemma 5.1 is true because the probability that a packet passes through a node n in level l is $1/2^l$. Every packet chooses its path independently and hence the probability that all of them pass through the node exactly equals $1/2^{lk}$. The rest of the proof is the same as before. Koch [9] has observed that the expected number of packets that get through is not affected by the rule that is used to decide which messages to keep and which messages to drop, as long as the destinations of the packets are not used to make this decision. Therefore, for this problem, random ranks are necessary only as a tool for analysis and any other non-predictive rule would exhibit the same average case behavior.

6 Open Questions

The most vexing problem left open by this paper is to determine the average number of time steps required to route on a fully loaded N-input butterfly with constant-size FIFO queues. If fewer than $\Omega(\log N)$ packets may be queued at a node, then the only known upper bound on the time to route is $O(N \log N)$. This trivial upper bound is proven by showing that after $\log N$ steps, at least one packet arrives at the outputs at every time step until the routing is completed. Simulations show that the true time is closer to $O(\log N)$.

Another open question concerns the algorithm of Section 4 for routing on a fully-loaded butterfly with constant size queues. We know from Section 2 that a single wave of packets with random destinations can be routed using a pure queuing protocol in $O(\log N)$ time, but when the waves are pipelined, as in Section 4, the analysis requires us to use a simple, but not pure, protocol to route each wave. It would be interesting to show that even if each individual wave was routed with a pure protocol, the total time to route $\log N$ waves was $O(\log N)$.

7 Acknowledgments

The authors would like to thank Danny Krizanc, Christos Kaklamanis, Tom Leighton, Satish Rao, Nick Pippenger, and Thanasis Tsantilas for many helpful discussions.

References

[1] W. A. Aiello, F. T. Leighton, B. M. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. *Mathematical Systems Theory*, 24(4):253–271, 1991.

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

[3] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, April 1979.

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

[5] Butterfly^{TM} Parallel Processor Overview. BBN Report No. 6148, Version 1, Cambridge, MA, March 1986.

[6] 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.

[7] A. Gottlieb. An overview of the NYU Ultracomputer Project. In J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 25–95. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987.

[8] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713–721, 1956.

[9] R. R. Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 221–230. IEEE Computer Society Press, October 1988.

[10] D. Krizanc. Oblivious routing with limited buffer capacity. Technical Report TR–14–87, Center for Research in Computing Technology, Harvard University, Cambridge, MA, 1987.

[11] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C–32(12):1091–1098, December 1983.

[12] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[13] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[14] 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 Computer Society Press, October 1988.

[15] T. Nakata, S. Matsushita, N. Tanabe, N. Kajihara, H. Onozuka, Y. Asano, and N. Koike. Parallel programming on Cenju: A multiprocessor system for modular circuit simulation. *NEC Research & Development*, 32(3):421–429, July 1991.

[16] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss. An introduction to the IBM Research Parallel Processor Prototype (RP3). In J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 123–140. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987.

[17] N. Pippenger. Parallel communication with limited buffers. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 127–136. IEEE Computer Society Press, October 1984.

[18] P. Raghavan. Lecture notes on randomized algorithms. Research Report RC 15340 (#68237), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, January 1990.

[19] A. G. Ranade. Constrained randomization for parallel communication. Technical Report YALEU/DCS/TR-511, Department of Computer Science, Yale University, New Haven, CT, 1987.

[20] A. G. Ranade. Equivalence of message scheduling algorithms for parallel communication. Technical Report YALE/DCS/TR-512, Department of Computer Science, Yale University, New Haven, CT, 1987.

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

[22] G. D. Stamoulis and J. N. Tsitsiklis. The efficiency of greedy routing in hypercubes and butterflies. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 248–259, July 1991.

[23] H. Suzuki, H. Nagano, T. Suzuki, T. Takeuichi, and S. Iwasaki. Output-buffer switch architecture for asynchronous transfer mode. In *Proceedings of the 1989 IEEE International Conference on Communications*, pages 99–103, June 1989.

[24] T. Szymanski and S. Shaikh. Markov chain analysis of packet-switched banyans with arbitrary switch sizes, queue sizes, link multiplicities and speedups. In *Proceedings of the IEEE INFOCOM '89*, pages 960–971, April 1989.

[25] A. M. Tsantilas. *Communication Issues in Parallel Computation.* PhD thesis, Harvard University, Cambridge, MA, 1990.

[26] E. Upfal. Efficient schemes for parallel communication. *Journal of the ACM*, 31(3):507–517, July 1984.

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

[28] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 943–971. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1990.

[29] 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.