Fast Algorithms for Routing Around Faults in Multibutterflies and Randomly-Wired Splitter Networks

F. Thomson Leighton[a,b] and Bruce M. Maggs[b]

Abstract–This paper describes simple deterministic $O(\log N)$-step algorithms for routing permutations of packets in multibutterflies and randomly-wired splitter networks. The algorithms are robust against faults (even in the worst case), and are efficient from a practical point of view. As a consequence, we find that the multibutterfly is an excellent candidate for a high-bandwidth low-diameter switching network underlying a shared-memory machine.

Index Terms–Fault tolerance, interconnection network, multibutterfly, multistage network, routing algorithm.

1 Introduction

Networks derived from hypercubes form the architectural basis of most parallel computers, including machines such as the BBN Butterfly, the Connection Machine, the IBM RP3 and GF11, the Intel iPSC, and the NCUBE. The butterfly, in particular, is quite popular, and has been demonstrated to perform reasonably well in practice. An example of an 8-input butterfly is illustrated in Figure 1. The nodes in this graph represent switches, and the edges represent wires. Each node in the network has a distinct label

[a]Mathematics Department and [b]Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. 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 OSR–89–0271, and the Army under Contract DAAL–03–86–K–0171. Tom Leighton was supported by an NSF Presidential Young Investigator Award with matching funds provided by IBM and AT & T. A preliminary version of this paper appeared in the *Proceedings of the 30th Annual Symposium on Foundations of Computer Science,* pages 384–389, October, 1989. Second author's current address: NEC Research Institute, Princeton NJ 08540

(r, l), where r is the row, and l is the level. In a butterfly with N inputs, the row is a $\log N$-bit binary number[1] and the level is an integer between 0 and $\log N$. The nodes on level 0 and $\log N$ are called the *inputs* and *outputs*, respectively. For $l < \log N$, a node labeled (r,l) is connected to nodes $(r, l+1)$ and $(r^l, l+1)$, where r^l denotes r with bit l complemented (bit 0 is the most significant, bit $\log N - 1$ the least).

The primary duty of the network in a parallel machine is to route messages between its processors and/or memory modules. In a butterfly, messages are typically sent from the switches on level 0, called the *inputs*, to those on level $\log N$, called the *outputs*. In a *one-to-one* routing problem, each input is the origin of at most one message, and each output is the destination of a most one message. One-to-one routing is also called *permutation routing*. All of the algorithms discussed in this paper route messages in a *store-and-forward* fashion. A store-and-forward algorithm treats messages as indivisible objects. At each step, a message can either remain at a switch or move in its entirety from one switch to another across an edge, provided that no other messages use the same edge at that step. Store-and-forward routing is also called *packet switching*, and messages are often referred to as packets. A related paper [2] extends the results of this paper for a different method of routing called *circuit switching*.

One of the nice features of the butterfly is that there is a simple algorithm for finding a path of length $\log N$ from any input to any output. Upon reaching switch (r, l), $l < \log N$, a packet with destination $(R, \log N)$ compares r with R. If they are equal, the packet takes the edge to $(r, l+1)$. If not, it takes the edge to $(r^l, l+1)$. One problem with the butterfly is that this path is unique. As a consequence, if some switch or edge along the unique path from input i to output j (say) becomes congested or fails, then communication between input i and output j will be disrupted.

1.1 Dilated butterflies

Because message congestion is a common occurrence in real networks, the wires in butterfly networks are typically *dilated*, so that each wire is replaced by a *channel* consisting of 2 or more wires. In a d-dilated butterfly, each channel consists of d wires. Because it is harder to congest a channel than it is to congest a single wire in a butterfly, dilated butterflies are better routing networks than simple butterflies [14, 16, 27].

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

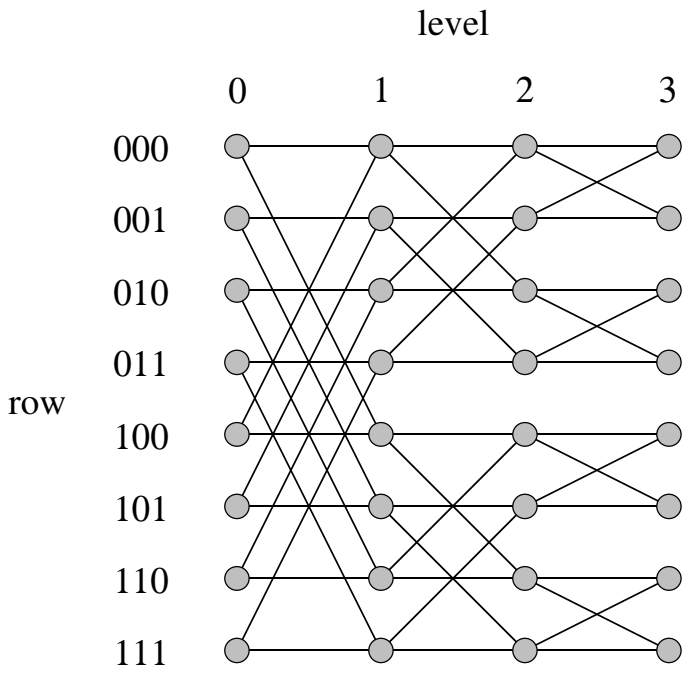

Figure 1: An 8-input butterfly network.

1.2 Splitter networks

Butterfly and dilated butterfly networks belong to a larger class of networks that are often referred to as *splitter networks*. The switches on each level of a splitter network can be partitioned into *blocks*. All of the switches on level 0 belong to the same block. On level 1, there are two blocks, one consisting of the switches that are in the upper $N/2$ rows, and the other consisting of the switches that are in the lower $N/2$ rows. In general, the switches in a block B of size $M = N/2^l$ on level l have neighbors in two blocks, B_u and B_l, on level $l+1$, where u stands for *upper* and l for *lower*. The upper block, B_u, contains the switches on level $l+1$ that are in the same rows as the upper $M/2$ switches of B. The lower block, B_l, consists of the switches that are in the same rows as the lower $M/2$ switches of B. The edges from B to B_u are called the *up* edges, and those from B to B_l are called the *down* edges. The three blocks, B, B_u, and B_l, and the edges between them are collectively called a *splitter*. The switches in B are called the *splitter inputs*, and those in B_u and B_l are called the *splitter outputs*. In a splitter network with *multiplicity* d, each splitter input is incident to d outgoing up edges and d outgoing down edges, and each splitter output is incident to

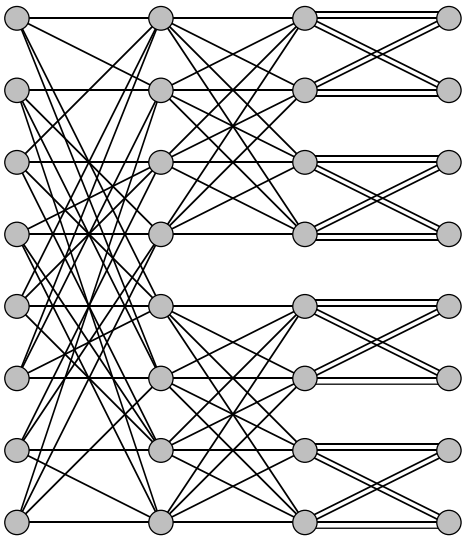

Figure 2: An 8-input splitter network with multiplicity 2.

$2d$ incoming edges. In a d-dilated butterfly, the d up (and d down) edges incident to each splitter input all lead to the same splitter output, but this need not be the case in general. For example, we have illustrated an 8-input splitter network with multiplicity 2 in Figure 2.

In a splitter network, each input and output are connected by a single logical (up-down) path through the blocks of the network. For example, Figure 3 shows the logical path from any input to output 011. In a butterfly, this logical path specifies a unique path through the network, since only one up and one down edge emanate from each switch. (In fact, a splitter network with multiplicity one is very similar to a delta network [17].) In a general splitter network with multiplicity d, however, each switch will have d up and d down edges, and each step of the logical path can be taken on any one of d edges. Hence, one logical path can be realized by a myriad of physical paths in a general splitter network.

1.3 Randomly-wired splitter networks and multibutterflies

In this paper, we are primarily concerned with randomly-wired splitter networks. A *randomly-wired splitter network* is a splitter network where the

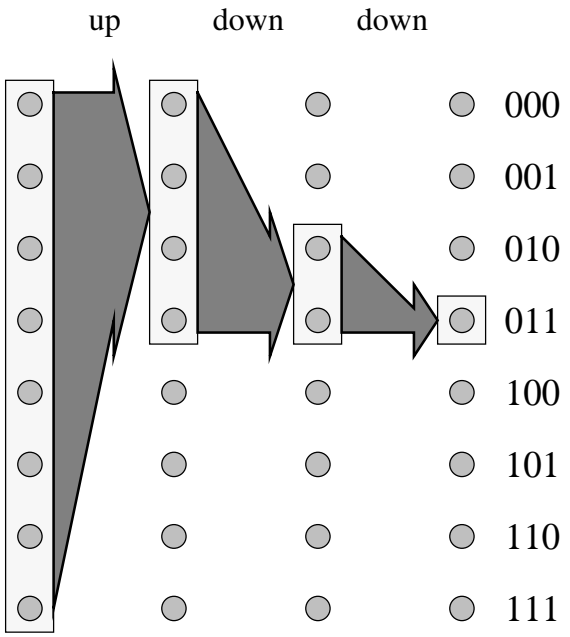

Figure 3: The logical path from any input to output 011.

up and down edges within each splitter are chosen at random subject to the constraint that each splitter input is incident to d up and d down edges, and each splitter output is incident to $2d$ incoming edges.

The crucial property that randomly-wired splitter networks are likely to possess is known as *expansion*. In particular, an M-input splitter is said to have (α, β)-expansion if every set of $k \leq \alpha M$ inputs is connected to at least βk up outputs and βk down outputs, where $\alpha > 0$ and $\beta > 1$ are fixed constants. For example, see Figure 4.

A splitter network is said to have (α, β)-*expansion* if all of its splitters have (α, β)-expansion. More simply, a splitter or a splitter network is said to have *expansion* if it has (α, β)-expansion for some constants $\alpha > 0$ and $\beta > 1$. A splitter network with expansion is more commonly known as a *multibutterfly* [29], and a *multibutterfly with* (α, β)*-expansion and multiplicity* d consists of splitters in which each splitter input is incident to d up and d down edges and for which any $k \leq \alpha M$ splitter inputs are adjacent to βk splitter outputs.

Splitters with expansion are known to exist for any $d \geq 3$, and they can be constructed deterministically in polynomial time [11, 23, 29], but ran-

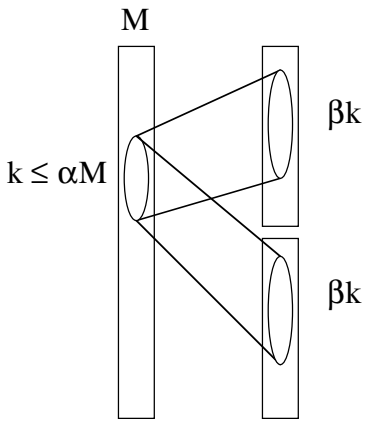

Figure 4: An M-input splitter with (α, β)-expansion.

domized wirings typically provide the best possible expansion. In fact, the expansion of a randomly-wired splitter will be close to $d-1$ with probability close to 1, provided that α is a sufficiently small constant. (For a discussion of the tradeoffs between α and β in randomly-wired splitters, see [20, 29].)

A multibutterfly with (α, β)-expansion is good at routing because one must block βk splitter outputs in order to block k splitter inputs. In classical networks such as the butterfly, the reverse is true: it is possible to block $2k$ inputs by blocking only k outputs. When this effect is compounded over several levels, the effect is dramatic. In a butterfly, a single fault can block 2^l switches l levels back, whereas in a multibutterfly, it takes β^l faults to block a single switch l levels back.

In 1989, Upfal showed that any N-input multibutterfly can route any one-to-one problem in $O(\log N)$ steps using a simple greedy algorithm [29], and that, by using pipelining, any N-input multibutterfly can route $O(\log N)$ one-to-one problems in $O(\log N)$ steps. Although the proof is complicated and the constants hidden by the Big Oh notation are large, the result is important because the only previously known deterministic on-line linear-hardware $O(\log N)$-step packet routing algorithm [18] requires the use of the AKS sorting circuit [1] (which is even more complicated and has even larger constant factors).

1.4 Our results

In this paper, we provide a substantially simpler analysis of the greedy routing algorithm on multibutterflies, and as a consequence, derive much smaller constant factors for the $O(\log N)$ time bound. The analysis uses a potential function argument that may eventually prove to be useful for other routing problems as well.

More importantly, we show that the multibutterfly is highly fault-tolerant. In particular, we prove that no matter how an adversary chooses f switches to fail, there will be at least $N - O(f)$ inputs and $N - O(f)$ outputs between which a simple variant of the greedy algorithm can route any $\log N$ permutations in $O(\log N)$ steps. Note that this is the best that we could hope for in general, since the adversary can always choose to isolate $\Omega(f)$ inputs and $\Omega(f)$ outputs by carefully selecting f faults. In the more commonly studied model of randomly located faults [10, 25, 26], we can do even better. For example, even if $\Theta(N \log N)$ faults are randomly-placed in the multibutterfly, with probability near 1, the network can still deterministically route a permutation on $\Theta(N)$ inputs and outputs. Thus the multibutterfly becomes the first bounded-degree network known to be able to sustain large numbers of faults with only minimal degradation in performance.

We also consider the experimental performance of the algorithms for $N = 1024$ inputs. Interestingly, we find that randomly-wired splitter networks with multiplicity 2 outperform 2-dilated butterflies with equivalent hardware on random routing problems, even if the splitter network has 100 faults. On worst-case routing problems such as transpose and bit reversal, the randomly-wired splitter networks perform substantially better than the dilated butterflies, which take $\Omega(\sqrt{N})$ steps. Hence, randomly-wired splitter networks appear to be good candidates for high-bandwidth, low-diameter switching networks underlying a shared-memory machine such as the BBN butterfly.

1.5 Related work

Randomly-wired splitter networks and multibutterflies have appeared in a number of different contexts. In 1974, Bassalygo and Pinsker [3] used (randomly-wired) splitter networks with expansion to construct the first nonblocking network of size $O(N \log N)$ and depth $O(\log N)$. In 1980, Fahlman [7] proposed a randomly-wired network called the *Hashnet*. In 1989, Upfal [29] studied splitter networks with expansion (which he called

multibutterflies - a term attributed to Ron Fagin) and described a simple deterministic algorithm for routing in $O(\log N)$ steps on an N-input multibutterfly. In another paper, Arora, Leighton, and Maggs [2] developed a circuit-switching algorithm for multibutterfly and *multi-Beneš* networks and showed that the algorithm can be used to establish connections in a nonblocking fashion. Most recently, DeHon, Knight, and Minsky [5] designed a 64-processor switching network using a randomly-wired splitter network for processor to memory communications. Experimental work on randomly-wired splitter networks is described in [4, 5, 13, 15, 21]. See also [6, 8] for other applications in which expanders are used to tolerate faults.

1.6 Outline

The remainder of the paper is divided into three sections as follows. The analysis of greedy routing algorithms on multibutterflies without faults is described in Section 2. In Section 3, we describe how to route around faults in $O(\log N)$ steps. The experimental data is presented in Section 4.

2 Routing without faults

In this section, we describe several algorithms for routing packets in $O(\log N)$ steps on an N-input multibutterfly. We start by describing and analyzing Upfal's greedy algorithm for routing P permutations in Sections 2.1 and 2.2. This duplicates the result of Upfal, except that the proof is simpler and the constants are better. In Section 2.3 we describe variations of the algorithm that allow pipelining and queueing for which the constant factors in the $O(\log N)$ running time bound are substantially smaller. We conclude in Section 2.4 by mentioning some improvements to the network itself.

Throughout this section, we will assume that, unless stated otherwise, the multibutterflies we are working with have multiplicity d and (α, β)-expansion, where $d \geq 3$, α is an integral power of $1/2$, and $\beta > 1$.

2.1 The algorithm

Upfal's greedy algorithm starts by partitioning the packets into *waves* so that at most one packet in each wave is destined for any set of L contiguous outputs, where L is a power of 2. One way to do this is to group packets into the same wave if they are in the same permutation and their destinations are congruent modulo L. If there are P permutations to be routed, this results

in the formation of at most PL waves. In general, we will set $L \geq \frac{1}{2\alpha}$ since then we will be guaranteed that at most $\left\lceil \frac{M}{2L} \right\rceil \leq \lceil \alpha M \rceil$ packets in any wave will ever pass through the up (or down) edges of any M-input splitter of the multibutterfly. Since M is a power of 2 and α is a power of $1/2$, αM is an integer unless $\alpha M < 1$, in which case $\lceil \alpha M \rceil = 1$. This will allow us to apply the (α, β)-expansion property to the set of inputs of any splitter occupied by the packets of a single wave at any time. (E.g., if k inputs of a splitter contain packets of a single wave that want to traverse up edges, then these inputs are connected to at least βk up outputs.) This is because packets going through the $\frac{M}{2}$ up (or $\frac{M}{2}$ down) splitter outputs can only be destined for the descendant set of $\frac{M}{2}$ contiguous multibutterfly outputs. For $\alpha M < 1$, an M-input splitter can receive at most one upward or downward destined packet, and we obtain expansion $d \geq \beta$. (Note that for $M/2 \leq d$, we can replace an M-input splitter with an $M \times M$ complete bipartite graph.)

The routing of the packets proceeds in stages, each stage consisting of an even and odd phase, and each phase consisting of $2d$ steps. In even phases, packets are sent from even levels to the next (odd) level, and in odd phases, packets are sent from the odd levels to the next (even) level. The edges within each splitter are colored with $2d$ colors so that exactly one edge of each color is incident on each input and exactly one edge of each color is incident on each output. In each phase, we process the colors in sequence, one step per color. For each color, we move a packet forward along an edge with that color if there is a packet in the switch at the tail of the edge that wants to go in that direction (up or down) and if there is no packet in the switch at the head of the edge. Alternatively, if there is a packet in the switch at the head of the edge and if it is in a later wave than the packet at the tail of the edge, then the two packets are swapped, so that the packet in the earlier wave moves forward. Note that every switch processes and/or contains at most one packet at any step.

If there is only one permutation to route, then each input of the multibutterfly starts with one packet. If there are P permutations to be routed, however, then it will be useful to augment the front-end of the multibutterfly with $P-1$ copies of the first level of the multibutterfly. Each switch on each of these levels starts with one packet. This will preserve the bound of queue size 1 at the input level, and it will ensure that these levels have the same (α, β)-expansion property as the first level. For notational purposes, we will refer to these additional levels as levels $-1, -2, \ldots, -(P-1)$.

2.2 The analysis

We will analyze the behavior of the greedy algorithm described in Section 2.1 by means of a potential function argument. In particular, we will assign each packet in wave i ($0 \leq i < LP$) *weight* ν^i for some fixed $\nu < 1$ to be determined later, and we will define the *potential of a switch* on level k ($-(P-1) \leq k \leq \log N$) to be w^k if k is odd, and w^{k+1} if k is even, where $w < 1$ is also a constant to be determined later. Thus, the *potential of a packet* in the ith wave at the kth level is $\nu^i w^k$ if k is odd, and $\nu^i w^{k+1}$ if k is even, and the *potential of the system* is the sum of potentials of the packets. By varying the values of ν and w, we can obtain different bounds on the running time of the algorithm as follows.

Theorem 2.1 *The algorithm of Section 2.1 routes P permutations in at most*

$$\frac{4d\log\frac{2}{w}}{\log\frac{1}{\rho}}\log N + \frac{4d\log\frac{1}{\nu}}{\log\frac{1}{\rho}}PL + \frac{4d\log\left(\frac{\nu}{L(1-\nu)(1-\nu^L w^{-1})}\right)}{\log\frac{1}{\rho}} = O(P+\log N)$$

steps, provided that $\beta > 1$ and ν is sufficiently small, where

$$\begin{aligned} L &\geq \frac{1}{2\alpha}, \\ \sigma &= \frac{\beta(1-\nu)}{1+\beta(1-\nu)}, \\ \rho &= [\sigma w + (1-\sigma)w^{-1}]^2. \end{aligned}$$

Corollary 2.2 *The algorithm in Section 2.1 routes a single permutation in at most*

$$\frac{4d\log\frac{2}{w}}{\log\frac{1}{\rho}}\log N + \frac{4dL\log\frac{1}{\nu}}{\log\frac{1}{\rho}} + \frac{4d\log\left(\frac{\nu}{L(1-\nu)(1-\nu^L w^{-1})}\right)}{\log\frac{1}{\rho}} = O(\log N)$$

steps.

Corollary 2.3 *The algorithm in Section 2.1 routes $\log N$ permutations in at most*

$$\frac{4d(\log\frac{2}{w} + L\log\frac{1}{\nu})}{\log\frac{1}{\rho}}\log N + \frac{4d\log\left(\frac{\nu}{L(1-\nu)(1-\nu^L w^{-1})}\right)}{\log\frac{1}{\rho}} = O(\log N)$$

steps.

The key fact necessary to prove Theorem 2.1 is that the potential of the system drops by a constant fraction during each stage. In particular, we will need to prove the following claim.

Claim 2.4 *After each stage of the algorithm, the total potential of the system decreases by a factor of at least*

$$\rho = [\sigma w + (1-\sigma)w^{-1}]^2$$

where

$$\sigma = \frac{\beta(1-\nu)}{1+\beta(1-\nu)}.$$

The main idea behind proving Claim 2.4 is that during any stage of the algorithm, a reasonable number of the heaviest packets move forward, thereby decreasing the potential of the system by a constant fraction. To formalize this intuition, we will need the following series of lemmas.

Lemma 2.5 *After any phase of routing packets through a splitter, the weight of a packet at the head of any edge cannot be less than the weight of the packet at the tail of the edge.*

Proof: Consider any edge e. At some step of the phase, the algorithm looks at the head and tail of e, and rearranges packets (if any) so that the heavier weight packet is sent to the head of e. In subsequent steps, only a packet with greater weight can be moved into the head of e, and only packets with lesser weight can move into the tail of e. Hence at the end of the phase, the weight of the packet at the head of e is at least as great as the weight of the packet at the tail of e, if any. (Nonexistent packets can be considered to have zero weight.) □

In what follows, let W_r^{ui} denote the sum of the weights of the r heaviest upward-destined packets left at the inputs of a splitter after a phase of routing through the splitter. If there are fewer than r upward-destined packets left at the inputs, then W_r^{ui} denotes the weight of all of them. A similar definition holds for W_r^{li} and also for W_r^{uo} and W_r^{lo}, which denote the sum of the weights of the r heaviest packets in the upper and lower outputs, respectively.

Lemma 2.6 *For any M-input splitter and any integer $r \leq \lceil \alpha M \rceil$, $W_{\lceil \beta r \rceil}^{uo} \geq \beta W_r^{ui}$ and $W_{\lceil \beta r \rceil}^{lo} \geq \beta W_r^{li}$.*

Proof: For simplicity, we will only argue the result for upward destined packets. The proof for lower-destined packets is identical.

We will prove by induction on r that $W^{uo}_{\lceil \beta r \rceil} \geq \frac{\lceil \beta r \rceil}{r} W^{ui}_r$. For $r = 1$, the result follows from Lemma 2.5 and the fact that every input is adjacent to $d \geq \beta$ upper outputs. Now assume the result is true for $r = k-1$, and look at the k heaviest upward-destined packets left at the inputs of the splitter. By the expansion property, we know that the inputs containing these packets are incident to at least $\lceil \beta k \rceil$ upper outputs. By Lemma 2.5, each of these outputs contains a packet of weight at least $W^{ui}_k - W^{ui}_{k-1}$ (i.e., the weight of the kth heaviest upper-destined packet left at an input). Hence, there are $\lceil \beta k \rceil$ or more upper outputs containing packets with weight at least $W^{ui}_k - W^{ui}_{k-1}$, and by induction, the $\lceil \beta(k-1) \rceil$ heaviest of these account for at least $\frac{\lceil \beta(k-1) \rceil}{k-1} W^{ui}_{k-1}$ weight. Thus,

$$\begin{aligned}
W^{uo}_{\lceil \beta k \rceil} &\geq \frac{\lceil \beta\,(k-1) \rceil}{k-1} W^{ui}_{k-1} + (\lceil \beta k \rceil - \lceil \beta\,(k-1) \rceil)(W^{ui}_k - W^{ui}_{k-1}) \\
&= (\lceil \beta k \rceil - \lceil \beta\,(k-1) \rceil)\, W^{ui}_k + \left(\lceil \beta\,(k-1) \rceil \frac{k}{k-1} - \lceil \beta k \rceil \right) W^{ui}_{k-1} \\
&\geq (\lceil \beta k \rceil - \lceil \beta\,(k-1) \rceil)\, W^{ui}_k + \left(\lceil \beta(k-1) \rceil \frac{k}{k-1} - \lceil \beta k \rceil \right) \frac{W^{ui}_k (k-1)}{k} \\
&= \frac{\lceil \beta k \rceil}{k} W^{ui}_k,
\end{aligned}$$

thereby verifying the inductive hypothesis. □

Lemma 2.7 *For $L \geq 1/2\alpha$, the total weights of all upper and lower-destined packets left at the inputs of any M-input splitter are at most $\frac{W^{ui}_{\lceil \alpha M \rceil}}{1-\nu}$ and $\frac{W^{li}_{\lceil \alpha M \rceil}}{1-\nu}$ respectively.*

Proof: For simplicity, we will only consider upward-destined packets. The same argument holds for lower-destined packets. Arrange the packets at the inputs in order of decreasing weight. Since $\lceil \alpha M \rceil \geq M/2L$, at most $\lceil \alpha M \rceil$ of these packets can belong to any wave. Hence the weight of the $(i + \lceil \alpha M \rceil)$th heaviest packet is at most ν times the weight of the ith heaviest packet for all i. Since the weight of the $\lceil \alpha M \rceil$ heaviest packets is $W^{ui}_{\lceil \alpha M \rceil}$ by definition, the total weight of all the upper-destined packets at the inputs is at most

$$W^{ui}_{\lceil \alpha M \rceil} + \nu W^{ui}_{\lceil \alpha M \rceil} + \nu^2 W^{ui}_{\lceil \alpha M \rceil} + \ldots \leq \frac{W^{ui}_{\lceil \alpha M \rceil}}{1-\nu},$$

as claimed. □

Lemma 2.8 *For $L \geq 1/2\alpha$, after one phase of routing packets through a splitter, a fraction of at least σ of all the packet weight in the splitter is at the outputs, where*

$$\sigma = \frac{\beta(1-\nu)}{1+\beta(1-\nu)}.$$

Proof: By Lemma 2.6, the total weight of packets on the upper and lower outputs is at least $\beta(W^{ui}_{\lceil \alpha M \rceil} + W^{li}_{\lceil \alpha M \rceil})$. By Lemma 2.7, the total weight of packets left at the inputs is at most $\frac{1}{1-\nu}(W^{ui}_{\lceil \alpha M \rceil} + W^{li}_{\lceil \alpha M \rceil})$. Hence the total weight on the outputs is at least $\beta(1-\nu)$ times the total weight on the inputs, and thus at least a $\frac{\beta(1-\nu)}{1+\beta(1-\nu)}$ fraction of the weight is on the outputs. □

Corollary 2.9 *After a phase of routing packets from level l to level $l+1$, at least a total fraction σ of the weight in the two levels is in level $l+1$.*

We are now ready to prove Claim 2.4 and Theorem 2.1.

Proof of Claim 2.4: Let x_l denote the weight at the beginning of the stage in levels l and $l+1$, where l is even. If V is the potential of the system at the beginning of the stage, then

$$V = \sum_{\text{even } l} x_l w^{l+1}.$$

During the first phase of the stage, packets move from even levels to odd levels. This does not change the potential of the system since each even level has the same potential as the next odd level. However, it does ensure that the weight in odd level $l+1$ is at least σx_l and the weight in even level l is at most $(1-\sigma)x_l$.

During the second phase, the weight in odd level $l+1$ is rearranged with the weight in even level $l+2$ for each l. By Corollary 2.9, and the argument of the previous paragraph, we know that at least $\sigma^2 x_l - (1-\sigma)^2 x_{l+2}$ weight moves from level $l+1$ to level $l+2$ for any l. Hence, the potential at the end of the stage is at most

$$\sum_{\text{even } l} [x_l - (\sigma^2 x_l - (1-\sigma)^2 x_{l+2}) + (\sigma^2 x_{l-2} - (1-\sigma)^2 x_l)] w^{l+1}$$

$$= \sum_{\text{even } l} x_l w^{l+1}[\sigma^2 w^2 + 2\sigma(1-\sigma) + (1-\sigma)^2 w^{-2}] = \rho V,$$

as claimed. □

Proof of Theorem 2.1: To compute an upper bound on the running time of the algorithm, we now need only to compute the initial potential of the system and the potential at which we will be guaranteed that every packet has reached its destination at level $\log N$.

To compute the initial potential, we will assume without loss of generality that the first L waves start in level 0, the next L waves start in level -1, and so forth. The total weight in the first L waves is at most

$$\frac{N}{L}(1 + \nu^1 + \nu^2 + \cdots + \nu^{L-1}) \leq \frac{N}{L(1-\nu)}.$$

Similarly the weight of the ith group ($1 \leq i \leq P$) of L waves is at most

$$\frac{N}{L}(\nu^{(i-1)L} + \nu^{(i-1)L+1} + \nu^{(i-1)L+2} + \cdots) \leq \frac{N\nu^{(i-1)L}}{L(1-\nu)}.$$

Hence the total initial potential is at most

$$\frac{N}{L(1-\nu)}\left(w + \frac{\nu^L}{w^1} + \frac{\nu^{2L}}{w^1} + \frac{\nu^{3L}}{w^3} + \frac{\nu^{4L}}{w^3} + \cdots\right) \leq \frac{N}{L(1-\nu)(1-\nu^L w^{-1})}.$$

The potential of a packet from the last wave on level $\log N$ is at most $\nu^{PL-1} w^{\log N}$. Hence, all of the packets must have reached level $\log N$ (their destinations) once the total potential falls below this amount. Since the total potential drops by a factor of ρ at every stage, the total number of stages can be at most S, where

$$\frac{N\rho^S}{L(1-\nu)(1-\nu^L w^{-1})} = \nu^{PL-1} w^{\log N}.$$

Hence, it suffices to choose

$$S = \frac{\log \frac{2}{w}}{\log \frac{1}{\rho}} \log N + \frac{\log \frac{1}{\nu}}{\log \frac{1}{\rho}} PL + \frac{\log\left(\frac{\nu}{L(1-\nu)(1-\nu^L w^{-1})}\right)}{\log \frac{1}{\rho}}.$$

Since each stage takes $4d$ steps, we can multiply by $4d$ to obtain the bounded stated in the theorem.

In order to show that $S = O(P + \log N)$, we need to show that there exist constants d, α, β, $w < 1$, and $\nu < 1$ such that $\nu^L/w < 1$ and $\rho < 1$. The key, of course, is showing that $\rho < 1$ (i.e., that the potential of the system drops during every stage). If $\sigma > 1/2$, then we can always find a value of $w < 1$ for which $\rho < 1$. (In particular, we can choose $w = \sqrt{(1/\sigma) - 1} < 1$, for then $\rho = 2\sigma(1 - \sigma) < 1$.) In order for σ to be greater than $1/2$, it suffices that $\beta(1 - \nu) > 1$. This can be accomplished for any $\beta > 1$ by setting ν to be sufficiently small. Making ν small also guarantees that $\nu^L/w < 1$. Hence, we only need to choose d, α, and β so that $\beta > 1$, which can be done for any $d \geq 3$ by choosing α to be sufficiently small.

For example, we could choose $d = 11$, $\alpha = 1/32$, $\beta = 4$, $L = 16$, $\nu = 1/2$, $\sigma = 2/3$, $w = 1/\sqrt{2}$, and $\rho = 8/9$, to show that $\log N$ permutations can be routed in at most $4273 \log N$ steps using the algorithm. □

2.3 Improving the constant factors

Although the constant factors derived for the running time bound in Theorem 2.1 are superior to those proved by Upfal [29] (who achieved a bound of $18500 \log N$ steps for $d = 21$), they are very far from what can be considered practical. Fortunately, the constant factors can be substantially improved if we allow each switch to process communications along each incident edge at the same time. In particular, we can handle $4d$ routing problems for the price of one by interleaving and pipelining the $4d$ problems in the standard way. (I.e., every edge will now be active at every step – edges with the jth color will be working on the ith problem during steps congruent to $i + j \bmod 4d$.) Similarly, we can partition a single problem into $4d$ *batches*, each of which can be handled simultaneously but independently. The number of waves in each batch is $PL/4d$ which means that we can simply reduce PL by a factor of $4d$ in the overall time bound. As long as $PL > 4d$, this means that we can route P permutations in

$$(4d \log \frac{2}{w} / \log \frac{1}{\rho}) \log N + (\log \frac{1}{\nu} / \log \frac{1}{\rho}) PL + 4d \log \left(\frac{\nu}{L(1-\nu)(1-\nu^L w^{-1})} \right) / \log \frac{1}{\rho}$$

steps.

The most interesting application of this procedure is to routing $\log N$ permutations, for which the time bound is $(4d \log \frac{2}{w} / \log \frac{1}{\rho} + L \log \frac{1}{\nu} / \log \frac{1}{\rho}) \log N + 4d \log \left(\frac{\nu}{L(1-\nu)(1-\nu^L w^{-1})} \right) / \log \frac{1}{\rho}$ steps. For $L = 16$, $\alpha = 1/32$, $\beta = 4$, $d = 11$, $w = 2/3$, $\nu = 1/4$, $\sigma = 3/4$, and $\rho = (\frac{7}{8})^2$, this gives an absolute bound of $367 \log N - 637$ steps which is substantially better than the previous bound.

Recently, Leighton and Maggs [19] developed an alternative method for analyzing greedy routing algorithms on multibutterflies which allows for another order-of-magnitude improvement in the constant factors. In particular, they have shown that a simple greedy algorithm can routing a single permutation in at most $56 \log N$ steps using $d = 10$ and queues of size 2 at each edge. By increasing d further, the time bound can be decreased to about $5 \log N$. Using a more complicated algorithm, and large d, the time bound can be decreased to nearly $2 \log N$ [2].

Even the best bounds for the constant factors are not very good, however. Fortunately, as we will see in Section 4, randomly-wired splitter networks (even those with $d = 2$) appear to perform much better in practice than would be indicated by the known theoretical upper bounds.

2.4 Other improvements

The algorithms described in Sections 2.1 through 2.3 can be improved in several ways. For example, by considering the expansion obtained across two or more levels of splitters, it is possible to obtain much larger expansion factors for randomly-wired splitter networks. In fact, it is possible to show that even a splitter network with multiplicity 2 can route any permutation in $O(\log N)$ steps. This is not possible to prove using the previous analysis for splitters with $d = 2$, since they do not have expansion.

One of the nice properties of a d-multibutterfly is that it requires about the same VLSI layout area [28] as a d-dilated butterfly. In particular, the layout area of an N-input d-butterfly or d-dilated butterfly is $\Theta(N^2 d^2)$. In Section 2.1 we proposed appending a set of levels numbered $-1, -2, \ldots, -(P-1)$ to the multibutterfly, each isomorphic to the first. Unfortunately, each such level requires $\Theta(N^2 d^2)$ area to lay out (as much as the entire multibutterfly), so the total area becomes $\Theta(PN^2 d^2)$. For $P \leq \log N + 1$, however, it can be shown that a more area-efficient network will suffice: the *multi-Beneš* network. The multi-Benes network consists of back-to-back multibutterflies and requires twice the area of the multibutterfly.

3 Routing around faults

In this section, we prove that the multibutterfly is a highly fault-tolerant network. We start by considering worst case faults in Section 3.1 and then consider the less malevolent case of random faults in Section 3.2. For simplicity, we will assume that $d \geq 5$, although similar results can be proved for

smaller d. On-line algorithms for reconfiguring around faults are discussed in Section 3.3.

3.1 Worst case faults

In this section, we will prove that no matter how an adversary selects f switches to be faulty, there are always at least $N - O(f)$ inputs and $N - O(f)$ outputs through which any permutation on those inputs and outputs can be routed in $O(\log N)$ steps. In fact, the greedy algorithm described in Section 2 works once we have removed an appropriate set of switches which include at most $O(f)$ inputs and outputs.

We first describe which outputs to remove. Examine each splitter in the multibutterfly and check if more than an ε fraction of the input switches are faulty, where $\varepsilon = 2\alpha(\beta' - 1)$ and $\beta' = \beta - \lfloor \frac{d}{2} \rfloor$. If so, then "erase" the splitter from the network as well as all descendant switches and outputs. The following lemma bounds the number of erased outputs.

Lemma 3.1 *The erasure process removes at most $f/\varepsilon = O(f)$ outputs.*

Proof: The erasure of an M-input splitter causes the removal of M multibutterfly outputs, and accounts for at least εM faults. Hence, at most $f/\varepsilon = f/2\alpha(\beta' - 1) = O(f)$ multibutterfly outputs are removed by this process. □

We next describe which inputs to remove. Working from level $\log N$ backwards, examine each switch to see if at least half of its upper output edges lead to faulty switches that have not been erased, or if at least half of its lower output edges lead to faulty switches that have not been erased. If so, then declare the switch to be faulty (but do not erase it). In the following lemmas, we will prove that at most $\frac{f}{\beta' - 1}$ additional switches are declared to be faulty at each level of this process. Hence, at most $O(f)$ multibutterfly inputs will be faulty (declared or otherwise).

Lemma 3.2 *In any splitter, at most a 2α fraction of the inputs are declared to be faulty as a consequence of propagating faults backwards. Moreover, at most an α fraction are propagated by faulty upper outputs and at most an α fraction are propagated by faulty lower inputs.*

Proof: The proof is by induction on the level, starting at level $\log N$ and working backwards. The base case at level $\log N$ is trivial since there are no propagated faults on level $\log N$. Now consider an arbitrary M-input splitter

on level i. Let U_i denote the set of faults propagated from the splitter's upper outputs and let L_i denote those propagated from the lower outputs. If the upper outputs were erased, then $|U_i| = 0$. Otherwise, the number of faults placed by the adversary in the upper outputs is at most $\varepsilon M/2$. Since each propagated input fault is connected to at most $\lfloor \frac{d}{2} \rfloor$ working upper outputs, we can conclude that $|U_{i+1}| + |L_{i+1}| + \varepsilon M/2 \geq |U_i|\,(\beta - \lfloor \frac{d}{2} \rfloor) = |U_i|\,\beta'$. By induction, $|U_{i+1}| \leq \alpha M/2$ and $|L_{i+1}| \leq \alpha M/2$. For $|U_i| > \alpha M$ and $\varepsilon = 2\alpha(\beta' - 1)$, we have a contradiction. A similar argument shows that $|L_i| \leq \alpha M$. □

Lemma 3.3 *Even if we allow the adversary to make f switches fail on every level, there will be at most $\frac{f}{\beta'-1}$ propagated faults on any level.*

Proof: The proof is again by induction on the level. Consider some level l and assume that it has more than $\frac{f}{\beta'-1}$ propagated faults. By Lemma 3.2, we know that these faults are divided among the splitters linking level l to $l+1$ so that we can apply the expansion property to the faults within each splitter. Hence there must be more than $\frac{(\beta - \lfloor \frac{d}{2} \rfloor)f}{\beta'-1}$ faults on level $l+1$. This is a contradiction however, since level $l+1$ can have at most $f + \frac{f}{\beta'-1} = \frac{\beta' f}{\beta'-1}$ total faults by induction. Hence, level l can have at most $\frac{f}{\beta'-1}$ propagated faults. □

We now erase all the remaining faulty switches. This leaves a network with $N - O(f)$ inputs and $N - O(f)$ outputs. Moreover, every input in every splitter is linked to $\lceil \frac{d}{2} \rceil$ functioning upper outputs (if the descendant multibutterfly outputs exist) and $\lceil \frac{d}{2} \rceil$ functioning lower outputs (if the corresponding multibutterfly outputs exist). Hence every splitter has an (α, β') expansion property. Thus, we can apply Theorem 2.1 with β replaced by β' to prove that the greedy algorithm still routes any permutation on the remaining inputs and outputs in $O(\log N)$ steps. This is summarized as Theorem 3.4 below. (Note that we can achieve expansion $\beta > \lfloor d/2 \rfloor + 1$ for $d \geq 5$.)

Theorem 3.4 *No matter which f interior switches are made faulty in an N-input multibutterfly, there are at least $N - \frac{f}{\beta'-1}$ inputs and $N - \frac{f}{2\alpha(\beta'-1)}$ outputs between which any $\log N$ permutations can be routed in at most $O(\log N)$ steps, provided that $\beta > \lfloor d/2 \rfloor + 1$.*

Proof: The results follows from Theorem 2.1, Lemmas 3.1 and 3.3, and the fact that $\beta' = \beta - \lfloor d/2 \rfloor > 1$. □

3.2 Random faults

Random faults are much easier to tolerate than worst-case faults. In fact, any network can be made to tolerate a large number of randomly placed faults simply by making multiple copies of its switches and edges. For example, suppose that an n-switch network G is replaced by a network G' in which for each switch u in G there are a pair of switches u and u' in G', and for each edge $\{u, v\}$ in G there are four edges $\{u, v\}$, $\{u, v'\}$, $\{u', v\}$, and $\{u', v'\}$. Then G' can simulate G provided that there is no pair of switches u and u' in G' that simultaneously fail. If each switch fails independently with probability $1/\sqrt{2n}$, then the expected number of failures is $\sqrt{2n}$, and the probability that any particular pair of switches u and u' both fail is $1/2n$. Since there are n pairs of switches, the probability that *any* pair fails is at most $1/2$. This technique is easily generalized. By making c copies of each switch and c^2 copies of each edge, any n-switch network can be made to tolerate a failure rate of $(cn)^{-1/c}$ with probability $1 - 1/c$.

By a similar argument, even if each switch in G' fails with some fixed constant probability then, with high probability, G' can simulate a constant fraction of the switches in G. In general, however, there is no guarantee that these switches in G will be connected in a useful fashion. As the following theorem shows, however, even if the switches fail with some constant probability, an N-input multibutterfly will have some set of $\Theta(N)$ inputs and $\Theta(N)$ outputs between which any permutation can be routed in $O(\log N)$ steps, with high probability. Furthermore, an N-input multi-Beneš network can tolerate constant failure probability and still route $\log N$ permutations of $\Theta(N)$ packets in $O(\log N)$ time. The only other networks that are known to tolerate constant failure probability are the N-switch hypercube, which can route any $\log N$ permutations of N packets in $O(\log N)$ time, with high probability, but has degree $\log N$ [10], and the N-switch mesh, which can route $\log N$ permutations of $\Theta(N)$ packets in $O(\sqrt{N} \log N)$ time [12].

The strategy for tolerating random faults is the same as the strategy of Section 3.1 for tolerating worst-case faults. We first examine each splitter to determine if more than an ε fraction of the input switches in that splitter are faulty, where $\varepsilon = 2\alpha(\beta' - 1)$ and $\beta' = \beta - \lfloor \frac{d}{2} \rfloor$. If so, then we erase the splitter and all of its descendant switches and outputs. Then we propagate faults back from level $\log N$ to level 0. A switch is declared to be faulty if at least half of its upper input edges or output edges lead to faulty switches that have not been erased. The following theorem shows that, with high probability, this strategy leaves a constant fraction of the network intact.

Theorem 3.5 *There exist fixed constants $p > 0$, and $\lambda > 0$, such that if each switch fails independently with probability p, then with probability at least $1 - e^{-\Theta(N/\log^2 N)}$ there is some set of λN inputs and λN outputs between which any permutation can be routed in $O(\log N)$ time.*

Proof: The hard part of the proof is showing that we don't erase too many outputs.

We begin by deriving an upper bound on the probability that more than εM input switches fail in an M-input splitter. Let S_M denote the number of input switches that fail in an M-input splitter. Then S_M has a binomial distribution, and

$$\Pr\{S_M \geq k\} \leq \binom{M}{k} p^k.$$

Setting $k = \varepsilon M$, we have $\Pr\{S_M \geq \varepsilon M\} \leq \binom{M}{\varepsilon M} p^{\varepsilon M}$. Using the inequality $\binom{a}{b} \leq (ae/b)^b$ and letting $\delta = \varepsilon(\ln(\varepsilon/p) - 1)$, we have $\Pr\{S_M \geq \varepsilon M\} \leq e^{-\delta M}$, where $\delta > 0$ for $p < \varepsilon/e$.

We can analyze the number of erased splitters on each level in a similar fashion. Consider a level of the network that contains N/M M-input splitters. Using the fact that each splitter on this level is erased independently with probability at most $e^{-\delta M}$, we can show that with probability at least $1 - e^{-\Theta(N/\log^2 N)}$, the number of erased splitters is at most $cN/M \log^2 M$ for any constant $c > 2\log^2 M e^{-\delta M}$. For small δ and all M, c can be made arbitrarily close to 0. Hence, with probability at least $1 - e^{-\Theta(N/\log^2 N)}$, at most $cN/\log^2 M$ outputs are erased due to faults occuring in splitters with M inputs. Summing over $M = 2, 4, 8, \ldots, N$, we find that with probability at last $1 - e^{-\Theta(N/\log^2 N)}$, at most $cN(1 + 1/4 + 1/9 + \cdots + 1/\log^2 N) \leq c\pi^2 N/6$ outputs are erased overall. Hence, at least λN outputs remain, where λ can be made close to 1 by setting δ to be close to zero.

Once all of the blocks containing more than an ε fraction of faulty switches are erased, we can apply Lemmas 3.2 and 3.3 to show that there aren't too many propagated faults on any level. In order to apply Lemma 3.3, we must bound the number of switches that fail on any level. To do this, we bound the binomial distribution to show that at most $c'N$ switches fail on a level with probability $1 - e^{\Theta(N)}$, where $c' < 2p$. By Lemma 3.3, if there are at most $c'N$ switch failures on any level, then there are at most $c'N/(\beta' - 1)$ propagated faults, and $\beta' c' N/(\beta' - 1)$ total faults on any level. Thus, with probability at least $1 - e^{-\Theta(N/\log^2 N)}$, some set of λN inputs and λN outputs remain, where $\lambda = \min\{1 - \beta' c'/(\beta' - 1), 1 - c\pi^2/6\}$. (Notice that $\lambda > 0$

since c and c' can be made arbitrarily small.) The time to route between these inputs and outputs is given by Theorem 3.4. □

3.3 On-line reconfiguration

Deciding which switches to remove from the network using the procedure described in Sections 3.1 and 3.2 is straightforward if we can reconfigure the network off-line. On-line reconfiguration is more challenging, however. The hard part is deciding locally which splitters contain too many faults at the input, since this calculation must be performed in the presence of faults. Fast on-line algorithms for this task are described by Goldberg, Plotkin, and Maggs in [9].

Once the splitters containing too many faults are erased, then it is a simple matter to propagate faults back through the network. Each switch simply checks its up and down output neighbors to see how many are faulty. As we will see in Section 4, the multibutterfly can tolerate a surprisingly large number of random faults without having to erase any inputs or outputs at all, so this simple fault propagation technique may be sufficient for reconfiguring multibutterflies in practice.

4 Experiments

This section presents data generated by a PASCAL program that simulates randomly-wired splitter networks. The experimental data demonstrates that a randomly-wired splitter network with multiplicity 2 is capable of routing efficiently even when many of its switches are faulty. The random numbers used in the simulations were generated using the minimal standard linear congruential generator from [24].

The program tested the ability of a randomly-wired splitter network with multiplicity 2 to tolerate randomly-placed faults. Four types of splitter networks were compared: normal butterflies, a 2-dilated butterfly, randomly-wired splitter networks with multiplicity 2, and randomly-wired splitter networks with multiplicity 2 that were modified to improve their fault tolerance. Each network had $N = 1024$ inputs. The randomly-wired networks were "cleaned up" after their creation to remove parallel edges between the same two switches where possible.

The program primarily simulated store-and-forward routing. Messages were routed according to a simple greedy algorithm. At each time step, a

10*	0.0
100*	0.0
250*	0.3
500*	1.3
750*	9.1
1000*	27.8

Table 1: Network failure rate. The left column shows the number of randomly-placed faults. The right column shows the percentage of trials in which at least one fault was propagated back to an input in a 1024-input modified randomly-wired splitter network with multiplicity 2. Each box represents 2000 trials.

switch was allowed to send a message along an edge provided that at the end of the previous step the number of messages in the switch at the end of the edge was at most 4. The average number of steps required to route all of the messages to their destinations was measured.

We made two modifications to the random networks in order to make them more fault tolerant. First, we removed the last 2 levels and replaced each 4-input splitter on these levels with a 4×4 complete bipartite graph. We then added a level numbered -1 with 4 random matchings to level 0. We use an asterisk (*) in Tables 1 through 3 to indicate that the networks were modified.

Faults were placed at random interior (i.e., not input or output) switches of the modified splitter networks. A non-faulty switch was declared faulty if either both of its up edges or both of its down edges led to faulty switches. Thus, faults could propagate from the last interior level back to the inputs. Because the faults were placed at random, the modified splitter networks could tolerate about $(N \log N)^{3/4}$ faults without any faults propagating back to the inputs. If any faults reached the inputs, then all of the faults were removed, and a new set of random faults was placed in the network. The number of faults in the modified splitter networks varied from 0 to 1000. Table 1 shows the percentage of trials in which at least one fault reached the inputs.

We first ran a set of trials to compare the performance of the randomly-wired splitter networks to the normal butterflies and the 2-dilated butterflies when routing random and worst-case problems. We ran 4 types of trials. The first type consisted of routing a message from each input to a random

	1	10	1T	10T
0 (nor), σ	14.1, 0.6	26.0, 1.0	38	272
0 (dil), σ	11.8, 0.4	18.7, 0.7	17	160
0, σ	11.1, 0.2	16.4, 0.5	11.8, 0.4	19.8, 0.5
0^*, σ	12.0, 0.3	18.0, 0.6	11.8, 0.4	17.2, 0.4
1^*, σ	12.0, 0.3	18.0, 0.6	11.8, 0.4	17.4, 0.6
10^*, σ	12.0, 0.3	18.3, 0.7	12.0, 0.5	18.4, 0.8
100^*, σ	12.2, 0.4	20.1, 1.3	12.7, 0.6	20.6, 1.3
250^*, σ	12.4, 0.5	21.8, 1.6	13.3, 0.7	22.7, 1.4
500^*, σ	12.9, 0.6	24.7, 3.0	14.0, 0.8	25.7, 2.2
750^*, σ	13.1, 0.7	26.6, 4.0	14.5, 1.3	28.2, 4.7
1000^*, σ	13.1, 1.0	26.5, 7.7	14.0, 1.9	27.5, 8.8

Table 2: Store-and-forward completion time. Each box shows the average, over 500 trials, of the number of steps for all of the packets to reach their destinations, and the standard deviation.

destination, for a total of $N = 1024$ messages. The second consisted of routing a collection of 10 such random routing problems. The third type was the transpose permutation, a worst-case problem for the butterfly and dilated butterfly. In this problem the destination of each message is formed by rotating the binary representation of the origin of the message $\frac{1}{2}\log N$ positions in a circular fashion. The last type consisted of 10 of these transpose permutations. For each of these types, we varied the number of faults in the modified randomly-wired splitter networks from 0 to 1000.

The data is presented in Table 2. There is one column in the table for each type of routing problem: a random problem (1), 10 random problems (10), a transpose problem (1T), and 10 transpose problems (10T). For each type of network tested, there is a row in the table. Each entry in a row shows the average, over 500 trials, of the number of steps required to route all of the packets to their destinations, and the standard deviation, σ. The butterfly rows are labeled 0 (nor), the 2-dilated butterfly rows are labeled 0 (dil), the randomly-wired splitter network rows are labeled 0, and the modified randomly-wired splitter network rows are labeled 0^* through 1000^* depending on the number of faults in the network.

Surprisingly, a randomly-wired splitter network with up to 100 faults performs nearly as well as the fault-free 2-dilated butterfly on random problems, and much better on transpose problems, even though both networks consist of the same amount of hardware.

	1	1T
0 (nor), σ	44.8, 1.1	3.1
0 (dil), σ	87.0, 1.0	12.5
0, σ	94.1, 0.7	89.9 , 0.8
0^*, σ	88.5, 0.9	89.9, 0.9
1^*, σ	88.5, 0.9	89.8, 0.9
10^*, σ	88.4, 0.9	89.6, 0.9
100^*, σ	86.5, 1.0	86.9, 1.0
250^*, σ	83.4, 1.1	82.5, 1.2
500^*, σ	77.9, 1.5	75.9, 2.2
750^*, σ	73.7, 5.1	71.4, 6.5
1000^*, σ	74.3, 11.2	73.4, 13.4

Table 3: Average throughput. Each box shows the average, over 500 trials, percentage of messages that successfully locked down their paths, and the standard deviation.

It is also possible to use this program to simulate circuit-switching. In the *circuit-switching* model, the goal of a message is to establish a dedicated path from its source to its destination. This path must be disjoint from the paths of all other messages; an edge can appear on at most one message's path. This model is most appropriate when the messages being transmitted are too long to be considered atomic objects. For circuit-switching we ran the same algorithm but instead measured the percentage of messages that reached their destinations without ever being delayed. These messages can be viewed as having successfully locked down paths from their sources to their destinations. The data is presented in Table 3. Here we found that a modified randomly-wired splitter network with 100 faults outperformed a fault-free 2-dilated butterfly! In a related paper [2], we describe more sophisticated circuit-switching algorithms that guarantee that all of the messages succeed in establishing their paths in $O(\log N)$ time.

Additional experimental data on the performance of randomly-wired splitter networks can be found in [4, 5, 13, 15, 21].

5 Remarks

The fault tolerance and potential function arguments developed in this paper can be applied to other leveled switching networks with local expansion properties. A good example of such a network is a fat-tree [22] with

expander-based switches. Using the methods described in this paper, it is possible to devise algorithms for routing around faults in this network, although the problem of optimally assigning packets to waves in a deterministic on-line fashion is still unresolved.

6 Acknowledgments

Thanks to Michael Ben-Or, Fred Chong, André DeHon, Andrew Goldberg, Michelangelo Grigni, Joe Kilian, Tom Knight, Serge Plotkin, Satish Rao, John Rompel, and Eli Upfal for numerous helpful comments.

References

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

[2] 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.

[3] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

[4] F. Chong, E. Egozy, and A. DeHon. Fault tolerance and performance of multipath multistage interconnection networks. In T. F. Knight, Jr. and J. Savage, editors, *Advanced Research in VLSI: Proceedings of the MIT/Brown Conference 1992*. MIT Press, March 1992. To appear.

[5] A. DeHon, T. F. Knight Jr., and H. Minsky. Fault-tolerant design for multistage routing networks. In *Proceedings of the International Symposium on Shared Memory Multiprocessing*, pages 60–71. Information Processing Society of Japan, April 1991.

[6] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Byzantine agreement in faulty networks. *SIAM Journal on Computing*, 17(5):975–988, 1988.

[7] S. E. Fahlman. The hashnet interconnection scheme. Technical Report CMU–CS–80–125, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, June 1980.

[8] J. Friedman and N. Pippenger. Expanding graphs contain all small trees. *Combinatorica*, 7(1):71–76, 1987.

[9] A. V. Goldberg, B. M. Maggs, and S. A. Plotkin. A parallel algorithm for reconfiguring a multibutterfly network with faulty switches. *IEEE Transactions on Computers*, 43(3):321–326, March 1994.

[10] 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.

[11] N. Kahale. Better expansion for Ramanujan graphs. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 398–404. IEEE Computer Society Press, October 1991.

[12] C. Kaklamanis, A. R. Karlin, F. T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. Thomborson, and A. Tsantilas. Asymptotically tight bounds for computing with faulty arrays of processors. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 285–296. IEEE Computer Society Press, October 1990.

[13] T. F. Knight, Jr. Technologies for low latency interconnection switches. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 351–358, June 1989.

[14] 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.

[15] S. Konstantinidou and E. Upfal. Experimental comparison of multistage networks. IBM Almaden Research Center. Unpublished manuscript, 1991.

[16] 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.

[17] C. P. Kruskal and M. Snir. A unified theory of interconnection network structure. *Theoretical Computer Science*, 48:75–94, 1986.

[18] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C–34(4):344–354, April 1985.

[19] F. T. Leighton and B. M. Maggs. Introduction to parallel algorithms and architectures: Expanders • PRAMs • VLSI. Manuscript in preparation.

[20] T. Leighton, C. L. Leiserson, and M. Klugerman. Theory of parallel and VLSI computation. Research Seminar Series Report MIT/LCS/RSS 10, MIT Laboratory for Computer Science, May 1991.

[21] T. Leighton, D. Lisinski, and B. Maggs. Empirical evaluation of randomly-wired multistage networks. In *Proceedings of the 1990 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 380–385. IEEE Computer Society Press, September 1990.

[22] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C–34(10):892–901, October 1985.

[23] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[24] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, October 1988.

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

[26] P. Raghavan. Robust algorithms for packet routing in a mesh. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 344–350, June 1989.

[27] R. D. Rettberg, W. R. Crowther, P. P. Carvey, and R. S. Tomlinson. The monarch parallel processor hardware design. *Computer*, 23(4):18–30, April 1990.

[28] C. D. Thompson. *A Complexity Theory for VLSI*. PhD thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[29] 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.

Frank Thomson Leighton (M'81) received the B.S.E. degree in electrical engineering and computer science from Princeton University in 1978, and the Ph.D. degree in applied mathematics from the Massachusetts Institute of Technology in 1981. He was among the first group of scientists to receive the NSF Presidential Young Investigator award. Currently he is a Professor of Applied Mathematics at the Massachusetts Institute of Technology, and a member of the MIT Laboratory for Computer Science. His research interests include parallel algorithms and architectures, discrete algorithms, VLSI, complexity theory, and combinatorics. He is the editor-in-chief of the Journal of the ACM and also serves on the editorial boards of SIAM Journal on Computing, SIAM Journal on Discrete Mathematics, Journal on Graph Theory, Journal of Algorithms, Cominatorica, and Algorithmica.

Bruce Maggs received the S.B., S.M., and Ph.D., degrees in computer science from the Massachusetts Institute of Technology in 1985, 1986, and 1989, respectively. Currently he is a Research Scientist at the NEC Research Institute in Princeton, NJ. His research interests include parallel computing systems and parallel algorithms.