Routing on Butterfly Networks with Random Faults

Richard Cole[1]
Courant Institute
New York University
New York, NY 10012
cole@cs.nyu.edu

Bruce Maggs[2]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
bmm@cs.cmu.edu

Ramesh Sitaraman[3]
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
ramesh@cs.umass.edu

Abstract

In this paper we show that even if every node or edge in an N-node butterfly network fails independently with some constant probability, p, it is still possible to identify a set of $\Theta(N)$ nodes between which packets can be routed in any permutation in $O(\log N)$ steps, with high probability. Although the analysis is complicated, the routing algorithm itself is relatively simple.

1 Introduction

This paper studies the ability of a network called a *butterfly* to route packets when many of its nodes and edges fail at random. A 32-node butterfly is shown in Figure 1. This network has been studied extensively and it, or one of its variants, has served as the routing network in several parallel computers. More information about the structural and algorithmic properties of butterflies can be found in the book by Leighton [13]. Some of the parallel computers that use butterfly networks are described in [7, 10, 18, 21, 24].

The algorithm described in this paper routes packets in a store-and-forward fashion. The term *store-and-forward* means that each non-faulty node has a queue in which it can store packets and, at each time step, can transmit at most one packet across each of its non-faulty edges. We will be primarily interested in routing packets between nodes in a *one-to-one* fashion, i.e., each node will be the source of at most one packet and the destination of at most one packet. One-to-one packet routing problems are also called *permutation* routing problems.

Throughout this paper, we use the following terminology to describe butterfly networks. A $\log n$-dimensional butterfly has $N = n(\log n + 1)$ nodes arranged in $\log n + 1$ *levels* of n nodes each. (Throughout this paper we use $\log n$ to denote $\log_2 n$.) Each node has a distinct label $\langle w, i\rangle$ where i is the level of the node ($0 \le i \le \log n$) and w is a $\log n$-bit binary number that denotes the *column* of the node. All nodes of the form $\langle w, i\rangle$, $0 \le i \le \log n$, are said to belong to column w. Two nodes $\langle w, i\rangle$ and $\langle w', i'\rangle$ are linked by an edge if $i' = i + 1$ and either w and w' are identical or w and w' differ only in the bit in position i'. (The bit positions are numbered 1 through $\log n$.) We call the first type of edge a *straight edge* and the second a *cross edge*. The nodes on level 0 are called the *inputs* of the network, and the nodes on level $\log n$ are called the *outputs*. Level 0 is the *top* of the butterfly, and level $\log n$ is the *bottom*. Sometimes the level 0 and $\log n$ nodes in each column are assumed to be the same node. In this case, the butterfly is said to *wrap around*. Our results hold whether or not the butterfly wraps around.

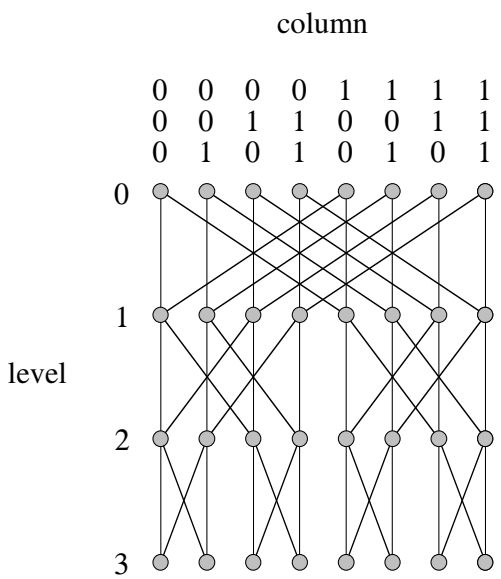

Figure 1: A 32-node butterfly network.

Our fault model is static. We assume that all faults occur before the network is used for routing and that all faults can be detected. We will use information about the locations of all of the faults to configure the network for routing. We assume that faults do not occur while the network is being used for routing.

[1]Richard Cole is supported in part by NSF Grants No. CCR-92-02900 and CCR-95-03309. [2]Bruce Maggs is supported in part by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute, and by ARPA Contract F33615-93-1-1330. [3]Ramesh Sitaraman is supported in part by NSF Grant No. CCR-94-10077.

Finally, we assume that a faulty edge cannot transmit a packet, but otherwise it does not affect either of its endpoints, and that a faulty node cannot transmit or receive packets.

1.1 Previous work

Many algorithms have been devised for routing packets on butterfly networks without faults. One of the earliest results is due to Beneš, who showed that it is possible to establish edge-disjoint paths between the nodes in the first and last levels of two back-to-back butterflies in any permutation [8]. Waksman [35] then gave an elegant algorithm for finding the paths. As a consequence of these two results, for any one-to-one packet routing problem on an N-node butterfly, it is possible to find, in $O(N \log N)$ time, a set of paths for the N packets and a schedule for moving the packets along their paths that delivers all of the packets to their destinations in $O(\log N)$ time steps. One drawback of Waksman's algorithm is that it uses global information about the network to find the paths for the packets. The next important advance was made by Batcher [6], who showed how to sort $\Theta(N/\log N)$ packets on an N-node butterfly in $O(\log^2 N)$ steps, using only local information. This algorithm can also be used for permutation routing, and it was the state of the art in permutation routing on butterfly networks for a long time. Next, Nassimi and Sahni [19, 20] showed how to route a wide class of permutations called bit-permute-complement permutations in $O(\log N)$ time. Finally, Valiant [33] showed how to route $\Theta(N/\log N)$ packets in any permutation in $O(\log N)$ steps. His idea was to first route each packet to a random intermediate destination. Improvements on Valiant's algorithm soon followed. First, Upfal [32] showed how to route N packets on an N-node butterfly network in $O(\log N)$ time. At about the same time, Aleliunas [4] showed how to do the same thing on an N-node shuffle-exchange graph. Algorithms for routing on butterflies with bounded-size queues were then devised by Pippenger [25], Ranade [28], and Maggs and Sitaraman [17]. Ranade also showed how combining could be used to solve not only one-to-one routing problems, but also many-to-one routing problems in $O(\log N)$ steps. As a consequence of Ranade's algorithm, it is possible for a butterfly network to emulate each step of a CRCW PRAM in $O(\log N)$ steps.

There is a vast literature devoted to the subject of routing on butterfly networks, or related multistage networks, with faulty components. For example, the query "fault!! AND network! AND (banyan OR butterfly OR omega OR multi!stage)," matches two hundred twenty-one records in the INSPEC database. Most of these papers can be placed into one or more of the following categories:

1. papers that suggest adding edges to the network so that there are multiple paths between any input and any output,
2. papers that suggest adding an extra stage to the network in order to tolerate a single stuck-at fault,
3. papers that suggest that packets can avoid faults by making multiple passes through the network,
4. papers that suggest using only a subset of the inputs and outputs for routing, and
5. papers that explore the problem of testing a network for faults.

Some of the earliest papers in these five categories are [23], [2, 3, 22, 30], [29], [5, 9], and [22] respectively. This paper falls into categories three and four. For a survey, see [1]. More recently, Varma [34] suggested that packets can avoid faults and still reach their destinations by first routing to an intermediate destination. Leighton, Maggs, and Sitaraman [15] proved that even if an adversary is allowed to place $\Theta(N/\log N)$ worst-case faults in an N-node butterfly network, it is still possible to identify some set of $\Theta(N)$ nodes between which packets can be routed in any permutation in $O(\log N)$ steps. They also showed that even if every node fails with some constant probability, the network can emulate a fault-free butterfly with $2^{O(\log^* N)}$ slowdown, with high probability. One consequence of this result is that it is possible to identify $\Theta(N)$ nodes between which packets can be routed in any permutation in $2^{O(\log^* N)} \log N$ steps. At about the same time, Tamaki [31] devised a simpler emulation scheme with $(\log\log N)^k$ slowdown, where k is some constant. The result translates into an algorithm for routing in $\log N(\log\log N)^k$ steps. Karlin, Nelson, and Tamaki [12] then showed that there is a critical failure probability, p^*, such that if every edge fails with probability p, and $p < p^*$, then it is very likely that in the subgraph of non-faulty nodes there exists a connected component of size $\Theta(N)$, but if $p > p^*$, it is very unlikely. Another approach worth mentioning is the "Information Dispersal" technique due to Rabin [26] and Lyuu [16]. They suggest using error-correcting codes to tolerate faults in routing networks. This approach works best when each packet is at least $\log^2 N$ bits long (and thus can be broken into $\log N$ smaller pieces), and the node failure probability is at most $1/\log N$.

1.2 Our results

In this paper we show that even if every node or edge in an N-node butterfly network fails independently with some constant probability, p, it is still possible to identify a set of $\Theta(N)$ nodes between which packets can be routed in any permutation in $O(\log N)$ steps, with high probability. In the course of analyzing the algorithm we also show that, even if there are random faults in the network, with high probability it is possible to establish constant-congestion fault-free paths in some one-to-one pattern from $\Theta(n)$ inputs to $\Theta(n)$ outputs.

1.3 Outline

The remainder of this paper consists of three sections. In Section 2, we show that, with high probability, it is possible to identify a set of $\Theta(n)$ inputs and $\Theta(n)$ outputs such that it is possible to route constant-congestion fault-free paths from the inputs to the outputs in a one-to-one fashion. This is the most difficult technical result in the paper. In Section 3, we show how to use the result of Section 2 to

find a set of $\Theta(N)$ nodes between which packets can be routed in any permutation in $O(\log N)$ steps. We conclude in Section 4 with some open problems.

2 Routing paths from inputs to outputs

In this section we show that, with high probability, it is possible to identify a set of $\Theta(n)$ inputs and $\Theta(n)$ outputs and a set of constant-congestion fault-free paths of length $\log n$ that connect the inputs to the outputs in a one-to-one fashion. We prove this result by showing that if each input is the source of at most one unit of flow, and each output is the sink of at most one unit, and each edge has constant integral capacity, then with high probability, it is possible to find a unidirectional integral flow of value $\Theta(n)$ from the inputs to the outputs.

In the remainder of this section, we assume without loss of generality that only nodes fail. Because the butterfly network has constant degree (4), it is not difficult to translate our result to the case in which edges fail as well.

A first attempt at finding a flow that avoids faults might proceed as follows. Identify inputs that can reach many outputs (say, at least as many as an input can reach on average) and outputs that can be reached from many inputs. Now for each input, split a unit of flow uniformly among all the outputs. Unfortunately, this does not necessarily result in a constant congestion flow. Thus, one would like to adjust the splits to achieve constant congestion while maintaining the total flow. It is far from clear that this can be done, or how to do it.

We achieve this in two steps. First, by adding additional faults, we are able to impose a very uniform structure on the faults. It then becomes possible to systematically adjust the splits of the flows so as to achieve a modest constant bound on the congestion. In addition, we show that the flow is unidirectional: it proceeds from the inputs of the network to the outputs without any backtracking. We will use this property when designing our algorithm for scheduling the movements of the packets along paths determined by the flow.

The fault structure has the following form: the network is edge-partitioned into strips of $\log d$ levels, i.e., successive strips share one level of vertices, for a suitable constant $d > 0$. (We mainly consider the case $d = 4$.) Each strip divides naturally into d-input subbutterflies. Any subbutterfly with a fault is not used further and is declared faulty. Furthermore, if any d-input subbutterfly S shares two or more inputs, or two or more outputs, with faulty subbutterflies in the previous and next strips, respectively, then S is declared faulty also. This rule is repeated until no more subbutterflies are declared faulty.

It is not hard to see that for $d = 4$, $O(n^{1/2})$ faults suffice to cause faults to propagate to all d-input subbutterflies in the network. With more effort, one can show that $\Omega(n^{1/2})$ faults are needed to cause faults to propagate to the whole network. But there are $\Theta(n \log n)$ random faults. To prove that $\Theta(n \log n)$ d-input subbutterflies remain fault-free, one has to exploit the fact that the faults are random. This is a non-trivial construction.

We will now impose further restrictions on the pattern of faults for the resulting fault-free d-input subbutterflies. The ideal situation would be complete uniformity, in the following sense: each non-faulty d-input subbutterfly would have exactly one faulty input neighbor and one faulty output neighbor (i.e., not zero). We cannot achieve this much. Instead, a more complex, but fairly uniform, structure is created (it is more complex in the sense that the neighborhood structure of collections of d-input subbutterflies are fully uniform as opposed to that of single d-input subbutterflies).

We can then specify a uniform routing in the sense that each input divides its unit of flow uniformly among the outputs it can reach. This routing involves bounded backtracking of the following form. The routing of the flow through one level of d-input subbutterflies may require using the preceding and following two levels of subbutterflies. Fortunately, the portion of the flow using the preceding and following levels of the butterfly uses only small (fractional) capacity. Consequently, it is possible to adjust the flows, by splitting the flow at nodes non-uniformly, in order to avoid the need for backtracking.

Finally, as the flow has constant congestion, with a unit bound at each of the inputs and outputs, we conclude that there is a constant congestion integral flow from the inputs to the outputs. This amounts to a collection of fault-free paths between paired inputs and outputs with the same constant congestion (arbitrary pairings are not possible, but there is a least one feasible collection of pairings).

Theorem 2.1 *For any constants $k_1 > 0$, $k_2 < 1$, and $\gamma \geq 1$ there is a constant $p > 0$ such that if every node fails with probability p, then with probability at least $1 - 1/N^{k_1}$ there is a set of $k_2 n$ inputs and $k_2 n$ outputs and a set of fault-free paths of length $\log n$ with congestion γ that connect the inputs to the outputs in a one-to-one fashion.*

Sketch of proof:
Part 1: We show that the following fault structure can be created by adding faults. Edge partition the butterfly into d-input subbutterflies, for some constant d. Each non-faulty d-input subbutterfly will have at least $d - 1$ non-faulty neighboring d-input subbutterflies on both the input and output sides; indeed, the faulty neighborhood structure is more constrained, as specified in part (iv). A statement of the results to be shown in each part follows.

(i) Use the following rule for propagating faults. Consider an edge partition of the butterfly into 4-input subbutterflies. If a subbutterfly has a faulty node view it as faulty. Any subbutterfly with two faulty neighbors on the input side or on the output side is itself viewed as faulty. Then it requires $n^{1/2}/2$ faults to make all the 4-input subbutterflies in the butterfly faulty.

Henceforth, we consider a fault as meaning a faulty 4-input subbutterfly. Note that a single faulty node may make two 4-input subbutterflies faulty, in the event that they share the node.

Lemma 2.2 *c faults induce at most c^4 faults at any given level of the butterfly.*

Lemma 2.3 *Suppose the butterfly is edge partitioned into b-input subbutterflies, each of which is either faulty or not faulty, and that we use the above rule for propagating faults among b-input subbutterflies (2 faulty input or output neighbors cause propagation). Then it requires $2^{\log n/\log b}/2$ faulty b-input subbutterflies to make all the b-input subbutterflies in the butterfly faulty, and $c = 2^t$ faults cause at most b^{2t} induced faulty b-input subbutterflies at any level of the butterfly.*

(ii) Consider a butterfly edge-partitioned into b-input subbutterflies. Suppose there are at most $b^\epsilon/2$ faults in each subbutterfly, for some $\epsilon < 1/20$. We show that after propagating faults among 4-input subbutterflies, there are at most $b^{4\epsilon}/16$ faults at the middle two levels in each $b-$input subbutterfly.

(iii) We generalize the above result to hold simultaneously for $\log\log\log n$ partitions into different-sized subbutterflies, where faults in one partition may generate faults in another. We also show that with high probability the number of induced faults at any level of the butterfly is at most $k_2 n$ for some constant $k_2 < 1$. Furthermore, for a suitable constant d, a partition into d-input subbutterflies leaves each non-faulty d-input subbutterfly with at least $d-1$ non-faulty input and output d-input subbutterfly neighbors.

(iv) Consider a given level of d-input subbutterflies. Partition them into sets of d^2 subbutterflies, each set together with its input and output neighbors forming a d^3-input subbutterfly. Form a second partition into sets of d^4 subbutterflies, where each set, together with its input and output neighbors going two levels forms a d^5-input subbutterfly.

We will add faults, if necessary, to impose the following structure on the faults. Arrange each set of d^2 subbutterflies in a $d \times d$ array, where each row comprises a set of subbutterflies with the same input neighbors and each column a set with the same output neighbors. A set of size d^2 has three choices: no faulty subbutterflies, all faulty, or one or both of a row and column of faulty subbutterflies. In addition, consider a collection C of d sets of size d^2 which have their input neighbors in an equal-sized collection (d sets of size d^2) at the input level. Then either all of the subbutterflies in C are faulty, or exactly d rows among these d sets of size d^2 are faulty. In the latter case, the faulty rows occur in one of three ways:

(1) All in one set of size d^2.

(2) The same row in each set.

(3) Different rows in each set.

An analogous constraint holds for the columns with respect to output neighbors. In the partition of d^4-input subbutterflies, the number of faulty rows is d^2, which is also the number of faulty columns.

Next, we fill in some of the details.

Part 1 (i)

Definition 2.4 *An h-subbutterfly has 4^h inputs.*

Suppose there are f faulty 1-subbutterflies in the butterfly.
a. We show that the induced faults on the top level 1-subbutterflies can all be produced by f faults on the top level.
b. We show that to make every subbutterfly on the top level faulty, using only faults on the top level, requires $n^{1/2}/2$ faults. The result follows.

a. We show how to move faults up one level of subbutterflies in the following sense, without increasing the number of faults. Let level $l+1$ be the lowest level of 1-subbutterflies with faults and suppose there are g faults at level $l+1$ (the top level is level 0). Note that an n-input butterfly has $(\log n)/2$ levels of 1-subbutterflies. We show that these g faults can be replaced by g or fewer faults at level l, such that every induced fault at level l (and above) remains present. It may be that some induced faults at lower levels are removed, however.

Consider the subbutterfly networks with inputs at level l and incorporating only lower levels in the network. Consider a maximal such subbutterfly with 4^h input nodes at level l in which every 1-subbutterfly becomes faulty through fault inducing.

We rearrange the level $l+1$ faults in the above subbutterfly without changing the induced faults at level l so that all the faults are in just two of the four $(h-1)$-subbutterflies with inputs at level $l+1$, and further these faults are arranged in pairs so that each pair of faults at level $l+1$ can be replaced by a corresponding pair of faults at level l without altering the induced faults in any way.

Definition 2.5 *An i-block is an i-subbutterfly with inputs at level $l+1$.*

Definition 2.6 *The 4 aligned i-blocks are the i-subbutterflies obtained from an $(i+1)$-subbutterfly with inputs at level l.*

Consider the process of inducing faults. The only induced faults we are interested in here are faults induced on level l by faults on level $l+1$ (pass-up faults), faults induced on level $l+1$ by faults on level l or higher (pass-down faults), and faults induced on level $l+1$ by faults on level $l+1$ (fill-in faults). We view the fault

inducing as occurring in a series of phases. In each phase we have pass-up, then pass-down, then fill-in.

In turn, we rearrange the faults in 1-blocks, ..., h-blocks, while maintaining the following property. For each collection of 4 aligned i-blocks:

(i) At most 2 of the 4 aligned i-blocks have faults.

(ii) Each pass-up fault occurs in the same or an earlier phase than before.

(iii) The complete fill-in of the second aligned i-block to be filled in with faults occurs in the same phase or in an earlier phase than before.

Base case: 1-blocks. There is no point to more than 2 faulty aligned 1-blocks. Simply remove any additional faults. (i) holds and clearly the process of inducing faults is unchanged.

Inductive case. Inductively, there are at most 8 out of 16 aligned $(i-1)$-blocks with faults, and for each set of four aligned $(i-1)$-blocks, at most 2 are faulty. Move the faults in these 8 $(i-1)$-blocks into 2 i-blocks, by performing swaps among aligned $(i-1)$-blocks; clearly each pair of aligned faulty $(i-1)$-blocks remains aligned. In addition, consider the first 4 $(i-1)$-blocks to fill in with induced faults, choosing at most 2 from each i-block before the swapping. There must be 4 such $(i-1)$-blocks for otherwise the $(i+1)$-block could not be filled with induced faults. Perform the swapping so that these four blocks are arranged two per i-block. Clearly (i) holds. (ii) holds because the alignments are maintained. (iii) holds because the chosen $(i-1)$-blocks are distributed to ensure that the fill-in of the first two i-blocks completes when these 4 $(i-1)$-blocks fill-in.

When the fault rearranging is complete, perform one more step of rearrangement to make all the faulted blocks consistent in the following sense: Consider the blocks with inputs on level $l+1$ and outputs on level $\frac{1}{2}\log b$. These are k-blocks, where $k = \frac{1}{2}\log b - l - 1$. For each collection of 4 k-blocks with the same neighbors at level l, 2 k-blocks are chosen to contain all the faults, by moving maximal fault blocks, if need be, to aligned locations in the chosen k-blocks.

Now consider a maximal faulted j-block B (in the sense that the $(j+1)$-block to which it belongs is not faulted). Let B' be its partner j-block containing some faults. Suppose they have s and s' faults respectively.

If $s \geq s'$, then rearrange the faults in B to be s' faults aligned with the faults in B'. Clearly, all the pass-up faults occur no later than before (for B now has exactly the same fill-in as B' and hence any pass-up fault, which always uses a fault in B' and the aligned fault in B, can occur no later). Further, the faults in B and B' are aligned and so can be moved from level $l+1$ to level l without affecting the fill-in at level l.

If $s < s'$, rearrange the faults in B' to be s faults aligned with the faults in B. The same argument applies.

b. Consider a butterfly with faults on the top level only.

We redefine the term i-block here.

Definition 2.7 *An i-block is a subbutterfly with 4^i nodes on the top level of the butterfly.*

Claim 2.8 *The faults in an i-block cause either all the 1-subbutterflies at the bottom level of the i-block to be faulty or none of them to be faulty.*

Proof: The proof is by induction. The base case $i = 1$ is trivial. For $i > 1$, consider the $(i-1)$-blocks. If two of more have faults at the bottom, then the i-block has faults at all its bottom 1-subbutterflies. If only one $(i-1)$-block has faults at its bottom, then the i-block has no faults at its bottom. □

Claim 2.9 *$n^{1/2}/2$ faults suffice to make the top level of an n-input butterfly faulty.*

Proof: By induction, to induce faults at the bottom of an i-block requires 2^{i-1} faults in the i-block (straightforward).

Thus to make the whole top level faulty (and hence the whole butterfly faulty) requires $n^{1/2}/2$ faults at the top level. □

We would like to show the same result is true for any intermediate level, and as far as we can tell it is true, but we don't have a proof, hence the weaker claim of Lemma 2.2, that c faulty 1-subbutterflies in the butterfly create at most c^4 induced faults on any given level.

Proof of Lemma 2.2. We show there are at most c^4 induced faults on level l. We proceed in three steps.

Step 1. Move the faults above level l down to level l without increasing the number of faults or changing the induced faults on level l.

Step 2. Similarly, move the faults below level l to level l.

Step 3. Consider the faults now at level l and discard any other faults as they do not induce faults at level l. Take the butterfly and fold it over at level l, creating a network with inputs at level l and fan-out 4, rather than 2, at each level. When folding the butterfly over, nodes are mapped staying in the same column. More precisely, suppose the fold is at node-level j $(= 2l$, where the fold is at 1-butterfly level $l)$. Then the node at level $j+i$ is overlaid on the node at level $j-i$ in the same column. Edges are added as follows. Between levels $j-i$ and $j-i-1$, where before we would define edges by possibly flipping bit $j-i-1$, now we flip one or both of bits $j-i-1$ and $j+i$. This creates a fan-out 4 butterfly-like structure and as subgraphs it has the butterflies from above and below the jth level (or more strictly an isomorphic image of the butterfly below level j). Edge partition the resulting butterfly into 16-input subbutterflies. Assume that if there are 2 faults at the input or output neighbors of a subbutterfly this makes the subbutterfly faulty also. As in the previous construction, we need $n^{1/4}/2$ faults to cut an n-input subbutterfly, and generalizing slightly, c faults at the inputs induce at most c^4 faults at any level (recall

that a subbutterfly with inputs on level 0 is either cut entirely or has no faults at its bottom level). □

Part 1 (ii)

In turn we show (a) and (b), below.

a. Consider a b-input butterfly. Suppose there are $2^{\epsilon \log b}/2 = 2^t$ faults anywhere in the butterfly, plus another f faults on the output (bottom) level. If $\log f + t + 2 < \frac{1}{2}\log b$, then there are at most $2^{4\epsilon \log b}/16$ faults at the input (top) level of the butterfly. Really $2^{2\epsilon \log b}/4$, but only the weaker bound will apply in a more general setting needed later.

b. The stated result for Part 1 (ii): Consider a butterfly edge-partitioned into b-input subbutterflies. Suppose there are at most $b^{\epsilon}/2$ faults in each subbutterfly, where $\epsilon 1/ < 20$. We show that after propagating faults among 4-input subbutterflies, there are at most $b^{4\epsilon}/16$ faults at the middle two levels in each subbutterfly.

Definition 2.10 *A subbutterfly is faulted if all its 1-subbutterflies are made faulty by the fault inducing due to faults it contains.*

a. Suppose $f = 2^i$ (add faults if need be). Consider the maximal subbutterflies that are faulted by these f faults and whose output level is part of the output level of the b-input subbutterfly. For each such maximal subbutterfly, suppose that the faulting set of faults is of minimum size. This partitions the f faults into sets of size 2^h, $0 \leq h \leq i$, where each set of faults is faulting a 4^{h+1}-input subbutterfly (by Part 1 (i) and Claim 2.9). Consider the 4-ary tree structure induced by the subbutterflies of various sizes with output level on the output level of the butterfly and tree ancestry corresponding to subbutterfly containment. Trim the tree keeping only the following nodes: the leaves correspond to the maximal subbutterflies faulted by the f faults and the interior nodes are their ancestors. In addition, we keep all children of interior nodes.

First, we show how to rearrange the f faults so they are all in one faulted subbutterfly with 4^{i+1} inputs, without reducing the number of induced faults at the input level of the butterfly and in addition without removing any induced faults except possibly in the subbutterflies faulted by the f faults. Consider two smallest subbutterflies, S and S', faulted by sets of $f' = 2^{h'}$ faults on the base. (As $f = 2^i$, and the faults always occur in sets of size 2^h, $h \geq h'$, faulting 4^{h+1}-input subbutterflies, there must be two equal-sized smallest faulted subbutterflies, each containing an equal number of faults, unless all the faults lie in one faulted subbutterfly.) For each subbutterfly the associated faults are defined as follows. Consider its 3 siblings in the tree defined above. The associated faults are those faults in the subbutterflies corresponding to the sibling nodes. Let S be the subbutterfly with a smaller number g of associated faults. Swap the f' faults in S with the associated g' faults of the other subbutterfly S' in the following way. Extend S' to a faulted $4^{h'+2}$-input subbutterfly using $2f'$ faults (i.e., fault S' with f' faults, and with another f' faults fault another subbutterfly associated with a sibling node of the node associated with S'). In S, place g of the g' faults associated with S' so that they are aligned with the g faults associated with S.

Consider the pass-up faults for the level above S. Clearly these are unchanged (previously fill-in due to pass-down faults immediately matched with the induced faults in S to cause pass-up faults; now the same pattern of fill-in occurs in S as in its siblings, and hence the same pass-up faults occur). Thus, w.l.o.g. we can assume there is just one subbutterfly with 4^{i+1} inputs, containing $f = 2^i$ faults at the output level of the butterfly.

Consider the path in the tree from the subbutterfly containing f faults to the root. Consider the number of faults in the subbutterflies associated with siblings of the nodes on this path (following the rearrangement of the f faults at the output into one faulted subbutterfly). Going up the tree, let this number of faults be $n_1, n_2, \ldots, n_{\frac{1}{2}\log b - i - 1}$.

Definition 2.11 *The base faults are the f faults on the output level. The in-block faults are the remaining faults in the butterfly. The induced top faults are original faults plus the induced faults at the input (top) level of the butterfly.*

Claim 2.12 *Consider rearranging the in-block faults but with the proviso that all the previously induced top faults remain induced top faults. Suppose that for all such rearrangements, some of the base faults are needed in order to induce all the induced top faults. Then:*
(i) $n_{j+1} > \sum_{1 \leq h \leq j} n_h$.
(ii) $n_{j+2} \geq 2^j n_1 + 2^j$.
(iii) $t + i + 2 \geq \frac{1}{2}\log b$.

Proof of Claim, part (i). Let $k = j$ be the first j for which the claim is not true. We move the faults for sibling blocks below the kth node on the path so that all the pass-up faults at the kth level (and higher) are unchanged and remove all f faults at the output level, thus: Take n_{k+1} of the faults from the subbutterflies associated with the lower level sibling nodes and place them in the subbutterfly associated with the kth node on the path, so that these faults are aligned with faults in its sibling blocks. As argued previously, the pass-up faults are unchanged, which yields a contradiction, since we assumed that the base faults were needed. □

Proof of Claim, part (ii). Part (ii) follows by induction from part (i). □

Proof of Claim, part (iii). Clearly $n_1 \geq 1$. $n_1, n_2, ..., n_{(\frac{1}{2}\log b)-i-1}$ all exist. $n_{(\frac{1}{2}\log b)-i-1} \geq 2^{(\frac{1}{2}\log b)-i-2}$. Also $2^{\epsilon \log b}/2 = 2^t \geq n_1 + n_2 + ... + n_{(\frac{1}{2}\log b)-i-1} \geq 2^{(\frac{1}{2}\log b)-i-2}$. Thus $t + i + 2 \geq \frac{1}{2}\log b$. □

Corollary 2.13 *If $t + i + 2 < \frac{1}{2}\log b$, there is a rearrangement of the in-block faults, such that even if all f base faults are removed, the set of induced top faults is either unchanged or incremented (this refers not only to the size of the set, but to its membership – any node originally in the set remains in the set).*

Hence the faults at the input level can all be induced by $2^{\epsilon \log b}/2$ in-block faults, and by Part 1(i), they induce at most $2^{4\epsilon \log b}/16$ faults.

Proof of Part 1 (ii) b.

Consider two edge partitions of the n-input butterfly into b-input subbutterflies. In the first partition, the bottom level of subbutterflies and the full butterfly share the same output level. The second partition excludes the bottom and top $\frac{1}{2}\log b$ levels of the full butterfly. Thus two overlapping subbutterflies from the two partitions share $b^{1/2}$ nodes per level on their overlapping $\frac{1}{2}\log b$ levels.

We verify inductively that the number of induced faults on the input and output levels of each of the subbutterflies is at most $b^{4\epsilon}/16$. This implies (trivially) that the number of faults on a $b^{1/2}$ portion of the input or output level (shared with an overlapping subbutterfly in the other partition) is at most $b^{4\epsilon}/16$.

The inductive claim is that when moving faults from bottom to top in the network, if no more than $b^{4\epsilon}/16$ faults enter along the output (bottom) levels of $b^{1/2}$-input subbutterflies at a given level, then no more than $b^{4\epsilon}/16$ faults leave the middle level of the b-input subbutterflies, where the two sets of subbutterflies share the same output level. An analogous claim is made for moving faults down.

The base case is provided by the subbutterflies at the bottom and top of the butterfly.

For the inductive step, we use the construction of part (a) with $f = b^{4\epsilon}/16$, $i = 4\epsilon \log b - 4$, $t = \epsilon \log b - 1$, and "b"$= b^{1/2}$. Consider moving faults from the bottom to the top of a $b^{1/2}$-input subbutterfly. The faults at the bottom of the subbutterfly induce no faults at the top of the subbutterfly of height $\frac{1}{2}\log b$, which is the lower middle level of the subbutterfly of height $\log b$ having the same base, if $i + t + 2 < \frac{1}{2}\log$ "b", i.e., $5\epsilon \log b - 3 < 1/4 \log b$, and this is satisfied if $\epsilon < 1/20$.

A similar argument applies when moving faults from the top to the middle of the b-input subbutterfly. Thus the faults at the middle levels are all induced by faults within the subbutterfly, and by Part 1 (i), this is at most $b^{4\epsilon}/16$ faults.

We conclude that the claim regarding induced faults on the two boundaries of a subbutterfly do indeed hold.

Part 1 (iii)

First it is helpful to generalize the Part 1 (ii) results as follows. Instead of considering faulty 4-input subbutterflies, we consider an edge partition into c-input subbutterflies. Each of these subbutterflies is either faulty or non-faulty. Also, a c-input subbutterfly is made faulty if it has 2 faulty input neighboring c-subbutterflies or 2 faulty output neighboring c-subbutterflies. Suppose the butterfly is edge partitioned twice into b-input subbutterflies also, $b > c$, where the two partitions overlap as in the previous section. If there are at most $2^{\epsilon(\log b - \log c)}/2$ faults within each b-input subbutterfly, then on a middle level of these b-input subbutterflies, there are at most $2^{4\epsilon(\log b - \log c)}/16$ induced faults, for $\epsilon < 1/20$. (A fault refers to a faulty c-input subbutterfly here.)

The claims holds, for another way of considering the c-input subbutterflies is that they are single supernodes and the vertex degree of the butterfly has been increased. A fault will still be propagated if a node has 2 faulty input or 2 faulty output neighbors. (A larger number of faulty neighbors for propagation will result in better probabilities in the bounds, but does not affect the construction otherwise.)

We will be considering a series of partitions with subbutterflies of $d_1 = d, d_2, d_3, \ldots$ inputs, the d_i's to be specified later.

Let $c = d_i$, $b = d_{i+1}$.

We define a d_1-input subbutterfly to be faulty if it contains one or more faults. A b-input subbutterfly is faulty if it contains more than $f(b, c)$ faulty c-input subbutterflies, f a function to be defined.

For the purposes of analysis we discard all faulty subbutterflies wholly contained in a larger faulty subbutterfly.

In turn we consider the propagating effects of the remaining faulty subbutterflies with $d_1, d_2, \ldots$ inputs.

The situation is made more complicated because in general there are two partitions into c-input subbutterflies, each of which may contain faulty subbutterflies. We name the two partitions as follows: the partition including all the levels of the butterfly is called the *full* partition and the second partition is called the *shifted* partition. We use a very simple rule for propagating faults from the shifted partition: each c-input subbutterfly in the full butterfly overlapping a faulty c-input subbutterfly in the shifted subbutterfly is declared to be faulty. The same fault pattern is achieved by adding two faulty c-input subbutterflies (which then propagate) to the full partition, where these added subbutterflies overlap the top levels of the faulty subbutterfly in the shifted partition.

We associate each c-input subbutterfly S in the shifted partition with two c-input subbutterflies in the full partition, arbitrarily chosen, except that we ensure that each full partition subbutterfly has two associated subbutterflies in the shifted partition, with the exception of those subbutterflies at the lowest level, which have no associated subbutterflies. S's associated subbutterflies are the ones declared faulty if S is faulty itself.

We are now ready to define the function $f(b, c)$. A b-input subbutterfly is faulty if it contains more than $2^{\epsilon(\log b - \log c)-1}/2 - 1$ c-input subbutterflies in the full partition which are either faulty subbutterflies in the full partition or associated with faulty subbutterflies in the shifted partition.

We show that if each non-faulty c-input subbutterfly requires at least two more faulty input neighbors or two more faulty output neighbors in order to become faulty, when ignoring faulty e-input subbutterflies, $e \geq c$, then so do the non-faulty b-input subbutterflies, when ignoring faulty f-input subbutterflies, $f \geq b$. A c or b-input subbutterfly is said to become faulty if it propagates even one fault from either its top to its bottom boundary, or its bottom to its top boundary.

For the base case, the non-faulty d_1-input subbutterflies satisfy the condition by construction.

The inductive step proceeds as follows. First map each faulty c-input subbutterfly in the shifted partition to two c-input subbutterflies in the full partition. This creates at most $2^{\epsilon(\log b - \log c)}/2 - 2$ faulty c-input subbutterflies in the full partition. Now, view the c-input subbutterflies as supernodes and apply the claim from the start of this section (Part 1(iii)). This asserts that for faults to propagate through a (non-faulty) b-input subbutterfly, it must contain more than $2^{\epsilon(\log b - \log c)}/2$ faulty c-input subbutterflies. But by construction, it contains at most $2^{\epsilon(\log b - \log c)}/2 - 2$ faulty c-input subbutterflies. Hence, to become faulty, it requires at least two additional faulty input neighbors or two additional faulty output neighbors. These faulty neighbors are created by the introduction of the faulty b-input subbutterflies.

It remains to derive some probability bounds. Let $b = 2^{r^2}, c = 2^r$. We show that given a probability $p_c \leq 1/2^{2r^2}$ that a c-input subbutterfly is faulty, the probability p_b that a b-input subbutterfly is faulty is at most $1/2^{2r^4}$. This is shown by an induction on i. The base case, $i = 1$, has $c = d = d_1$ and $b = d_2$; it holds trivially by choosing p small enough. For the inductive step, it is helpful to consider collections of c-input butterflies, such that the subbutterflies in each collection fail independently. As we will see, for each b-input subbutterfly, there will be $4c$ collections. Then we sum the probabilities that one of these collections contains at least $1/4c$ of the number of faulty c-input subbutterflies causing the b-input subbutterfly to fail.

As a result of the remapping of faults for finer partitions, we claim that a c-input subbutterfly can be affected by faults it contains or by faults in overlapping subbutterflies of height $(\sum_{h=2}^{i-1} \log d_h)$ sharing half their height with the c-input subbutterfly. Another way of considering this is that a c-input subbutterfly's faultiness is determined by faults contained within a subbutterfly of height at most $(\sum_{h=2}^{i-1} \log d_h)/2 + d_i \leq 2(\log c - 1)$, which has the same top level as the c-input subbutterfly. To verify the claim inductively, we consider a b-input subbutterfly. Aside from the c-input subbutterflies it contains, it's faultiness may be affected by faulty overlapping c-input subbutterflies from the shifted partition. The faults making these overlapping subbutterflies faulty are contained in subbutterflies of height $(\sum_{h=2}^{i-1} \log d_h)/2 + d_i$, overlapping the b-input subbutterfly to height $(\log c)/2$. The claim now follows.

Consider two c-input subbutterflies in either the full or the shifted partition that are two levels apart in the partition. Clearly, they fail independently, for the two levels are height $\log c$ apart. The collections of subbutterflies are formed as follows. For each partition, separate the subbutterflies into two groups, comprising alternating levels of subbutterflies. For each group, separate it into c collections, where each set of c subbutterflies in a common c^2-input subbutterfly is distributed one per collection.

There are at most $br^2/(cr)$ c-input subbutterflies in the full partition of a b-input subbutterfly, and at most twice as many associated c-input subbutterflies from the shifted partition. Each partition is divided into $2c$ equal sized collections. Thus each collection holds at most $br^2/(c \cdot cr)$ c-input subbutterflies. For the b-input subbutterfly to fail, at least one of its collections must contain $(1/4c)2^{\epsilon(r^2-r)-2}$ faulty c-input subbutterflies. Thus:

$$
\begin{aligned}
p_b &\leq 4c \binom{br^2/(c \cdot cr)}{2^{\epsilon(r^2-r)-2-\log(4c)}} (1/2)^{2r^2 \cdot [2^{\epsilon(r^2-r)-2-\log(4c)}]} \\
&\leq 4 \cdot 2^r \cdot (2^{r^2} \cdot r/2^{2r})^{[2^{\epsilon(r^2-r)-4-r}]} / 2^{2r^2 \cdot [2^{\epsilon(r^2-r)-4-r}]} \\
&\leq (4r/2^r)^{2^{\epsilon(r^2-r)-4-r}} 1/2^{r^2 \cdot 2^{\epsilon(r^2-r)-4-r}} \\
&\quad \text{if } \epsilon(r^2-r) - 4 - r \geq 0 \ (r \text{ large enough}) \\
&\leq 1/2^{r^2 \cdot 2^{\epsilon(r^2-r)-4-r}} \\
&\quad \text{if } 2^r \geq 4r \ (r \geq 4)
\end{aligned}
$$

and the claim follows if $2^{\epsilon(r^2-r)-4-r} \geq 2r^2$. But this holds for large enough r.

Thus there are $\log\log\log n$ levels of partitions and the probability that the whole butterfly fails is only $1/2^{2\log^2 n} = 1/n^{2\log n}$.

As we propagate faults whenever there are two neighboring input or output faults each remaining non-faulty d-input subbutterfly has at most one input and one output neighbor that are faulty.

Part 1 (iv)

A difficulty in achieving the claimed distribution of faults arises if there is more than one faulty subbutterfly in a d^2 size set initially. To avoid this problem, we declare that the whole d^2-size set of d-input subbutterflies is faulty if it contains even a single faulty subbutterfly. However, sets of d^2 d-input subbutterflies can have one or more faulty d-input subbutterflies become faulty by propagation without the whole set of d^2 subbutterflies becoming faulty. By reducing p appropriately, the probability bounds of Part 1 (iii) are maintained.

Then the only fault patterns that can arise in a set of d^2 subbutterflies are: all faulty, none faulty, a row faulty, a column faulty, a row and column faulty.

We seek to achieve the fault pattern described below for each collection of d sets of d^2 subbutterflies, having d^3 d-input subbutterflies as their (shared) input neighbors. Either (1) or (2) are present, or a subset of (3) is present.

(1) All faulty rows are in one set of size d^2.

(2) The same faulty row is present in each set.

(3) Different faulty rows are present in each set.

That these are the only options can be seen as follows. The subbutterflies (input neighbors) in a column at one level are all connected to the subbutterflies in a row at the next level. For short, say the column is connected to the row. Each set of d^2 subbutterflies at one level, viewed as d columns, is connected to d rows at the next level. These d rows are spread over d collections of d^2 subbutterflies. It is convenient if these rows all have the same index, l say. Clearly if two of these index l rows are faulty, then the set of d^2 subbutterflies one level up is completely faulty, thereby making all the index l rows faulty also. This is case (2). The only other options are cases (1) and (3).

If case (3) applies, then case (3) also applies to the columns for the corresponding collection among the input neighbors. Thus it is safe to extend the faulty rows to a full permutation, in the sense that this does not induce any further fault propagation.

Part 2

Consider the rows and columns of d d-input subbutterflies, as defined in Part 1(iv). Either all the subbutterflies in a row/column are faulty or at most one of the subbutterflies is faulty. Say that the row/column is good in the latter case.

Each good row has for its input neighbors a good column and each good column has as its output neighbors a good row.

The number of good columns and good rows in a set of d^4 subbutterflies, as defined in Part 1(iv) is equal $(d^4 - d^2)$.

Next, we create a flow. The flow is straightforward: to go from one level of subbutterflies to the next, push the flow from a column to the associated row, with the same total flow going from and to each node. Then within a level, within a set of d^4 subbutterflies, push each row's flow to a distinct column. For those size d^2 sets of subbutterflies with the same number of faulty rows and columns this does not require any additional action. For the remaining subbutterflies, the two preceding and following levels need to be used in order to route this flow. This is constant congestion as d is a constant. Connectivity is present by inspection.

Next we adjust the flow so that there is no backing up, i.e., it is unidirectional.

Let us assume each row carries $d(d-1)$ units of flow (recall a row contains d d-input subbutterflies, of which at most one is faulty).

Let's consider the adjustments that are needed for one collection of d^4 subbutterflies to make the row to column transitions. The only problems arise with sets of d^2 subbutterflies that have unequal numbers of good rows and columns. Consider such a subbutterfly set S with one faulty row (and no faulty columns). We will show how to push 1 unit of flow from each non-faulty subbutterfly in S to a companion subbutterfly in a set S' of d^2 subbutterflies, where S' has one faulty column and no faulty rows.

Consider the $d \times d$ array formed by arranging the d^2 sets of d^2 subbutterflies, where a row contains those sets having common neighbors 2 levels up on the input side, and a column contains the sets with common neighbors on the output side. Permute the columns and rows so that the sets of d^2 subbutterflies that all fail are on the leading diagonal and form a topmost portion of the diagonal. Suppose there are r such sets. Then if S has index (i,j) and S' has index (k,l), we have $j \leq r < i$ and $k \leq r < l$. To route, we push the flow from S to S'' and then from S'' to S', where S'' has index (i,l) or (k,j). The pairings are chosen so that the S'' are unique and $k \neq l$ if S'' has index (k,j). A little thought (draw a picture) shows that such pairings are always possible.

We detail the case in which S'' has index (k,j); the other case is similar and is left to the reader. The flow from S to S'' uses the network for two levels below the current level and that from S'' to S' uses the two levels above the current level. We now show that the flows from the various S to S'' use independent paths. Consider one column of S. Every non-faulty subbutterfly in the column connects to every subbutterfly in the row one level below. We send a proportionate flow to each subbutterfly in the row from each subbutterfly in the column, which is either $1/d$ or $1/(d-1)$ units of flow per edge (depending on whether the row has zero or one faulty subbutterflies). This flow needs to be switched to another row in the set of size d^2 containing this row; the new row is the one adjacent to a column in set S''. This transfer, from one row to the other, is accomplished by going down one more level in the network. Again, use each of the d or $d-1$ non-faulty intermediate butterflies. The flow is either $(d-1)/d^2$, $1/d$ or $1/(d-1)$ units of flow.

Within a d-input subbutterfly, it is less clear what the flows are. But another way to consider such a subbutterfly is to consider the flow entering and leaving. Then it is simply a matter of pushing this flow from inputs to outputs. As there are no faults in the subbutterfly this is straightforward.

Altogether there are 5 adjustments to the flow at each level, totaling at most $5/(d-1)$. With d sufficiently large, there is no negative flow, i.e., all flow is one-way.

Some care is needed in considering what happens at the top two levels (resp. bottom two levels) of subbutterflies. The problem case arises at the second level when a set of d^2 subbutterflies has a faulty row but no faulty column (the opposite arrangement does not occur as it is always possible to add a faulty row, assuming the network does not wrap around). An easy solution is to have the initial faults at the first level propagate to the second level, making the adjacent set of d^2 subbutterflies faulty. For then the above problem case does not occur.

With wrap around, one solution is to adjust the flows at the input nodes to the network. Then scale the whole flow to bound the input at each node by one unit of flow.

Thus, for any constants $\gamma \geq 1$ and $k_2 < 1$, by making p small enough, we can route a flow with congestion γ between $k_2 n$ inputs and outputs. □

3 Routing around faults

In this section we use the result of Theorem 2.1 to derive an $O(\log N)$ time algorithm for routing packets between some set of $\Theta(N)$ nodes.

3.1 Routing the identity permutation

Let us begin by first considering the simpler problem of routing packets in a one-to-one fashion between the n inputs of the network. In order to do this, packets will have to travel back and forth through the network. Each packet will travel from an input to an output along a flow path, and then back from that output to its destination input along a (possibly different) flow path.

There is one permutation that we already know how to route in this manner: the identity permutation. According to Theorem 2.1, there is a set of $k_2 n$ inputs and $k_2 n$ outputs between which paths of length $\log n$ can be routed in a one-to-one fashion with constant congestion. In order to route the identity permutation, each packet first routes from its input to the corresponding output, and then routes back along the same path. Since the congestion is constant, each packet can be delayed for at most a constant number of steps at each switch, and since the paths have length $O(\log N)$, the total time is $O(\log N)$. (Of course, for this permutation, the packets could have just stayed in place, but bear with us.)

3.2 Routing more interesting permutations

What if we want to route more interesting permutations? Let π_l, $0 \leq l < n$, denote the permutation in which the input in column i sends a packet to the input whose binary number is $\text{bin}(i) \oplus \text{bin}(l)$, i.e., the column whose binary number is the exclusive-or of the binary representation of i and the binary representation of l. The identity permutation, for example, is π_0.

Suppose that there were no faults. Then we could route π_l by first routing each packet from its input of origin straight down its column to an output. On the way back up from the outputs to the inputs, the packet would take a straight edge from level j to $j-1$ if bit j in the binary representation of l is 0, and a cross edge otherwise. Note that, at every level, either all of the packets take straight edges, or all of the packets take cross edges.

Another way to understand this algorithm is to view the network as two back-to-back butterflies sharing output nodes. In order to route π_l, we first take the second butterfly and fold it back onto the first butterfly. At this point each node of the second butterfly lies above the corresponding node in the first butterfly, except for the output nodes, which are shared. We are now going to "nail down" nodes of the second butterfly onto nodes of the first butterfly, starting at the outputs and working our way one level at a time towards the inputs. In particular, when we reach the jth level of the second butterfly, $j > 0$, (the outputs are on level $\log n$, the inputs on level 0), we examine the jth bit in the binary representation of l. If this bit is 0, then we simply nail every node in level $j-1$ of the second butterfly onto the node in the first butterfly that lies below it. If however, the bit is 1, then we first exchange groups of $N/2^{j-1}$ columns in the second butterfly. In particular, we exchange the nodes in levels 0 through $j-1$ of the second butterfly in column i with the corresponding nodes in the column whose binary representation differs from $\text{bin}(i)$ in bit j, for $0 \leq i < n$. We then nail down the nodes in level $j-1$. When we reach the inputs ($j = 0$) of the second butterfly, we stop. This folding and exchanging algorithm maps the nodes of the second butterfly to the nodes of the first butterfly in a one-to-one fashion. Furthermore, if two nodes in the second butterfly are connected by an edge, then the nodes to which they have been nailed in the first butterfly are also connected by an edge. To route permutation π_l, we now route each packet down its column in the first butterfly, and then back up the same column in the second butterfly.

Now suppose that there are faults in the network, and that we want to route permutation π_l on a large subset of the inputs. We begin by nailing down the nodes of the second butterfly onto the first butterfly as described above. Next, we consider a node of the first butterfly to be faulty if either it or the node of the second butterfly that has been nailed to it is faulty. As a consequence, the failure probability of each node in the first butterfly will at most double. Now we apply Theorem 2.1 to find a set of paths from $k_2 n$ inputs to $k_2 n$ outputs in the first butterfly. (Note that because we have doubled the failure probability, we must divide the probability p given in the theorem by 2.) Now to route π_l on the $k_2 n$ inputs and outputs, we first route each packet along its flow path in the first butterfly, and then back along the same path in the second butterfly. Because the path is fault-free in the first butterfly, it must also be fault free in the second butterfly.

Because the probability of failure given by Theorem 2.1 is at most $1/N^{k_1}$, and there are only n different permutations π_l, the probability that the above algorithm fails for any permutation is at most n/N^{k_1}. Since $n < N$, this failure probability is at most $1/N^{k_3}$, where $k_3 = k_1 - 1$. Hence, with high probability (at least $1 - 1/N^{k_3}$), we can route any of the permutations π_l on a set of at least $k_2 n$ inputs.

3.3 Identifying inputs for packet routing

If we could route each permutation π_l on the same set of $k_2 n$ inputs, then we would use those inputs for routing. Unfortunately, the set of inputs may differ for each permutation. As a consequence, will not be able to identify a large set of inputs for which all of the permutations π_l can be routed using the preceding method. For large k_2, however, we will be able to identify a large number of inputs that succeed in routing a large fraction of the permutations π_l. In particular let rn be the number of inputs that succeed on at least 3/4 of the permutations. Since $k_2 n$ inputs succeed on every permutation, $rn + (3/4) \cdot (n - rn) \geq \cdot k_2 n$, so

$rn \geq 4(k_2 - 3/4)n$. Let $k_4 = 4(k_2 - 3/4)$. Then with probability at least $1 - 1/N^{k_3}$, we have identified a set of $k_4 n$ inputs, each of which can route to its correct destination for 3/4 of the permutations π_l.

3.4 The routing algorithm

We are now in a position to describe the packet routing algorithm. We begin by assuming that each of the $k_4 n$ inputs is the origin of $(\log n + 1)$ packets, and each is the destination of $(\log n + 1)$ packets. Later we will show how to route packets in any one-to-one pattern between some set of $\Theta(N)$ nodes in the network. We say that an input i can reach another input j if the path for permutation π_l, where $\text{bin}(l) = \text{bin}(i) \oplus \text{bin}(j)$, is fault free. A packet routes from its origin a to its destination b as follows. It begins by selecting at random an input c that both a and b can reach. Since both a and b can reach at least $(3/4)n$ inputs, there are at least $n/2$ choices for c. Input c will serve as a random intermediate destination. The packet routes to c using the path for permutation π_{l_1}, where $\text{bin}(l_1) = \text{bin}(a) \oplus \text{bin}(c)$. It then routes from c to b using b's path for permutation π_{l_2}, where $\text{bin}(l_2) = \text{bin}(b) \oplus \text{bin}(c)$. Since both π_{l_1} and π_{l_2} require two passes through the network, the packet ends up making a total of four passes.

Although we have described the algorithm for a single packet, in fact all of the packets proceed at once. Note that since each packet is selecting its own intermediate destination at random, the packets will be using paths from many different permutations π_l simultaneously.

3.5 Bounding the congestion

In order to bound the time for all of the packets to reach their destinations, we must first bound the congestion of the paths selected by the packets. The *congestion* of an edge in the network is the maximum number of packets that traverse the edge. The congestion of the network is the maximum edge congestion.

Lemma 3.1 *For any constant $k_6 > 0$, there is a constant $k_7 > 0$ such that with probability at least $1 - 1/N^{k_6}$, the congestion is at most $k_7 \log N$.*

Proof: We analyze only the congestion of the first half of the path of each packet, i.e., the path to its random intermediate destination. By symmetry the same bound holds on the congestion of the second half.

Each packet independently chooses a random intermediate destination, and uses its path from some permutation π_l to route to that destination. By Theorem 2.1, if every packet used the same permutation π_l, the congestion would be at most $2\gamma(\log n + 1)$ (recall that there are $(\log n + 1)$ packets at each input). The packets all use different permutations, however, and the probability that any particular permutation is chosen is at most $2/n$. Let e be an edge of the network and let $u_{i,l,e}$ be 1 if permutation π_l sends the ith packet $0 \leq i < n(\log n + 1)$ through edge e of the network, and 0 otherwise. The probability that packet i uses edge e, $p_{i,e}$, can thus be bounded as

$$p_{i,e} \leq \sum_{l=0}^{n-1} 2u_{i,l,e}/n.$$

Let c_e be the congestion of edge e. Then

$$\begin{aligned} E[c_e] &= \sum_{i=0}^{n(\log n+1)-1} p_{i,e} \\ &\leq \sum_{i=0}^{n(\log n+1)-1} \sum_{l=0}^{n-1} 2u_{i,l,e}/n \\ &= \frac{2}{n} \sum_{l=0}^{n-1} \sum_{i=0}^{n(\log n+1)-1} u_{i,l,e} \\ &\leq \frac{2}{n} \sum_{l=0}^{n-1} 2\gamma(\log n + 1) \\ &= 4\gamma(\log n + 1) \end{aligned}$$

Thus, the expected congestion is at most $O(\log n)$. We now use a lemma due to Hoeffding and a Chernoff-type bound to show that, with high probability, the congestion exceeds the expectation by at most a constant factor.

Hoeffding's lemma [11], stated below, says, essentially, that if X is the sum of a collection of independent 0-1 random variables, then for any particular $E[X]$, X is most likely to deviate from $E[X]$ when all of the 0-1 variables have the same expectation. Thus, if we want an upper bound on the probability that X deviates from $E[X]$ it suffices to consider this special case. A Chernoff-type bound can then be used to bound the probability that too many of the 0-1 variables have value 1.

Lemma 3.2 *Let X be the number of successes in r independent Bernoulli trials where the probability of success in the ith trial is q_i. Let S be the number of successes in r independent Bernoulli trials where each trial has probability of success $q = \frac{1}{r}\sum_{1 \leq i \leq r} q_i$. Then $E(X) = E(S) = rq$, and*

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

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

□

In our application, $X = c_e$, $E[X] \leq 4\gamma(\log n + 1)$, $r = n(\log n + 1)$, and $q_i = p_{i,e}$.

We will use the following Chernoff-type bound [27, p. 56].

Lemma 3.3 *Let S be the number of successes in r independent Bernoulli trials where each trial has probability q of success. Then $E(S) = rq$, and*

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

for $\alpha > 2e$.

□

Thus,

$$\begin{aligned}\Pr[c_e \geq 4\alpha\gamma(\log n+1)] &\leq \Pr[S \geq 4\alpha\gamma(\log n+1)] \\ &\leq 2^{-4\alpha\gamma(\log n+1)} \\ &\leq n^{-4\alpha\gamma}\end{aligned}$$

for $\alpha > 2e$. By choosing α to be a large enough constant, we can make this probability as small as $1/4N^{k_5}$, for any constant $k_5 > 0$.

Since there are fewer than $2N$ different edges in the entire network, the probability that any edge has congestion greater than $4\alpha\gamma \log N$ is at most $1/2N^{k_5-1}$. The same bound holds for the second half of the path of each packet. Hence, setting $k_7 = 8\alpha\gamma$ and $k_6 = k_5 - 1$ completes the proof. □

3.6 Scheduling the packets

Every routing algorithm performs two tasks: finding paths for the packets to take through the network, and scheduling the movements of the packets along their paths. By Lemma 3.1, with high probability, our algorithm will find paths for the packets with congestion $O(\log N)$, where each path makes four passes through the network and uses butterfly edges in one direction only during each pass.

Another way of viewing the routing algorithm is that the butterfly simulates a network with $4 \log N$ levels in which the paths pass from level 0 to level $4 \log N$ and in which every edge is directed from one level to the next. A network of this form is called a *leveled* network. It is convenient to view the butterfly this way because it allows us to apply an off-the-shelf scheduling algorithm for leveled networks. In particular, we can use the algorithm of Leighton, Maggs, Ranade, and Rao [14]. With high probability, this algorithm will route N packets in $O(c+L+\log N)$ steps, where c is the congestion and L is the depth (distance from first to last level) of the network. In our application, $c = O(\log N)$ and $L = O(\log N)$, so the total time is $O(\log N)$.

3.7 One-to-one routing

Thus far we have described an algorithm for routing $(\log n + 1)$ packets from each input and to each input. If, instead, we want to route packets in a one-to-one fashion between some set of $\Theta(N)$ nodes in the network, then we spread out the packets starting at each input along a path to some output. Consider the paths defined by the identity permutation. Clearly, we can route packets along these paths to and from the input nodes. Thus we use as input nodes those nodes that can route both the identity permutation and a sufficient fraction of all other permutations. This may require changing the 3/4 fractions to 7/8.

The performance of the algorithm described in this section is summarized by the following theorem.

Theorem 3.4 *For any constants $k_8 > 0$, $k_9 > 0$, and $k_{10} < 1$, there are constants $p > 0$ and $k_{11} > 0$ such that, even if every node or edge in the network fails independently with probability p, with probability at least $1 - 1/N^{k_8}$, it is possible to identify a set of $k_{10}N$ nodes between which packets can be routed in any one-to-one pattern in $k_{11} \log N$ steps (with probability at least $1 - 1/N^{k_9}$).*

4 Open problems

Although this paper shows that it is possible to route in $O(\log N)$ time on a butterfly with constant-probability random faults, it is still not known whether such a butterfly can emulate a fault-free butterfly with constant slowdown. The slowdowns of the emulations of Leighton, Maggs, and Sitaraman [15] and Tamaki [31] are $2^{O(\log^* N)}$ and $\log\log^k N$ (where k is some fixed constant), respectively. Both functions grow very slowly with N, but are not constant.

Another open problem concerns the ability of the butterfly (or Beneš) network to route constant-congestion paths between some set of $\Theta(n)$ inputs and outputs in any permutation. The results in this paper imply that even if every node fails with some constant probability, it is possible to route paths between inputs and outputs, but the congestion will be $\Theta(\log N/\log\log N)$. For worst-case faults, Leighton, Maggs, and Sitaraman [15] showed that a butterfly (or Beneš) network can tolerate up to $n^{1-\epsilon}$ faults, where ϵ is any fixed constant greater than zero, and still route any permutation on some set of $n(1-o(1))$ inputs and outputs with constant congestion. Whether the network can tolerate more than $n^{1-\epsilon}$ faults and still route constant-congestion paths remains open.

References

[1] G. B. Adams, III, D. P. Agrawal, and H. J. Siegel. A survey and comparison of fault-tolerant multistage interconnection networks. *Computer*, 20:14–27, June 1987.

[2] G. B. Adams, III and H. J. Siegel. The extra stage cube: A fault-tolerant interconnection network for supersystems. *IEEE Transactions on Computers*, C–31(5):443–454, May 1982.

[3] Dharma P. Agrawal. Testing and fault tolerance of multistage interconnection networks. *Computer*, pages 41–53, April 1982.

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

[5] J. Arlat and J. C. Laprie. Performance-related dependability evaluation of supercomputer systems. In *Proceedings of the 13th Annual International Symposium on Fault-Tolerant Computing*, pages 276–283, June 1983.

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

[7] Butterfly™ Parallel Processor Overview. BBN Report No. 6148, Version 1, BBN Advanced Computers, Inc., Cambridge, MA, March 1986.

[8] V. E. Beneš. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, July 1964.

[9] C. R. Das and L. N. Bhuyan. Reliability simulation of multiprocessor systems. In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 591–598, August 1985.

[10] 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.

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

[12] A. R. Karlin, G. Nelson, and H. Tamaki. On the fault tolerance of the butterfly. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 125–133, May 1994.

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

[14] 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.

[15] T. Leighton, B. Maggs, and R. Sitaraman. On the fault tolerance of some popular bounded-degree networks. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 542–552, October 1992.

[16] Y.–D. Lyuu. *Information Dispersal and Parallel Computation*. Cambridge University Press, Cambridge, England, 1992.

[17] B. M. Maggs and R. K. Sitaraman. Simple algorithms for routing on butterfly networks with bounded queues. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 150–161, May 1992.

[18] 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.

[19] D. Nassimi and S. Sahni. A self-routing benes network and parallel permutation algorithms. *IEEE Transactions on Computers*, C–30(5):332–340, May 1981.

[20] D. Nassimi and S. Sahni. Optimal BPC permutations on a cube connected computer. *IEEE Transactions on Computers*, C–31(4):338–341, April 1982.

[21] W. Oed. Cray Y-MP C90: system features and early benchmark results. *Parallel Computing*, 18(8):947–954, August 1992.

[22] D. C. Opferman and N. T. Tsao-Wu. On a class of rearrangeable switching networks–part II: Enumeration studies and fault diagnosis. *Bell System Technical Journal*, 50(5):1601–1618, May–June 1971.

[23] K. Padmanabhan and D. H. Lawrie. A class of redundant path multistage interconnection networks. *IEEE Transactions on Computers*, pages 1099–1108, December 1983.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] J. P. Shen and J. P. Hayes. Fault-tolerance of dynamic-full-access interconnection networks. *IEEE Transactions on Computers*, C–33(3):241–248, March 1984.

[30] S. Sowrirajan and S. M. Reddy. A design for fault-tolerant full connection networks. In *Proceedings of the International Conference on Science and Systems*, pages 536–540, March 1980.

[31] H. Tamaki. Efficient self-embedding of butterfly networks with random faults. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, October 1992. 533-541.

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

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

[34] A. Varma. Fault-tolerant routing in unique-path multistage interconnection networks. *Information Processing Letters*, 31(4):197–201, May 1989.

[35] A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, January 1968.