Randomly-Wired Multistage Networks

Bruce M. Maggs

NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

Randomly-wired multistage networks have recently been shown to outperform traditional multistage networks in three respects. First, they have fast deterministic packet-switching and circuit-switching algorithms for routing permutations. Second, they are nonblocking, and there are on-line algorithms for establishing new connections in them, even if many requests for connections are made simultaneously. Finally, and perhaps most importantly, they are highly fault tolerant.

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 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 N-input butterfly ($N = 8$) with depth $\log N = 3$ is shown in Figure 1. The nodes in this graph represent switches, and the edges represent wires. Messages are typically sent from the switches on level 0, called the *inputs*, to those on level $\log N$, called the *outputs*.

The message routing algorithm for a butterfly is quite simple. Each message simply follows the unique path of length $\log N$ from its source input to its destination output. One problem with this algorithm (and hence the network) is that 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.

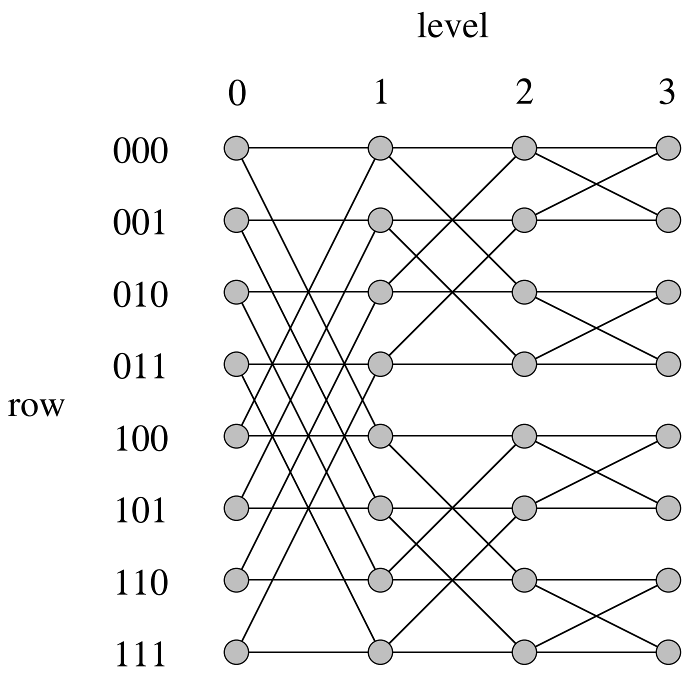

Figure 1: An 8-input butterfly network.

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 [8, 9, 16].

1.2 Delta networks

Butterfly and dilated butterfly networks belong to a larger class of networks called *delta* networks [10]. The switches on each level of a delta 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 on level l have neighbors in two blocks, B_u and B_l, on level $l+1$. The upper block, B_u, contains the switches on level $l+1$ that are in the same rows as

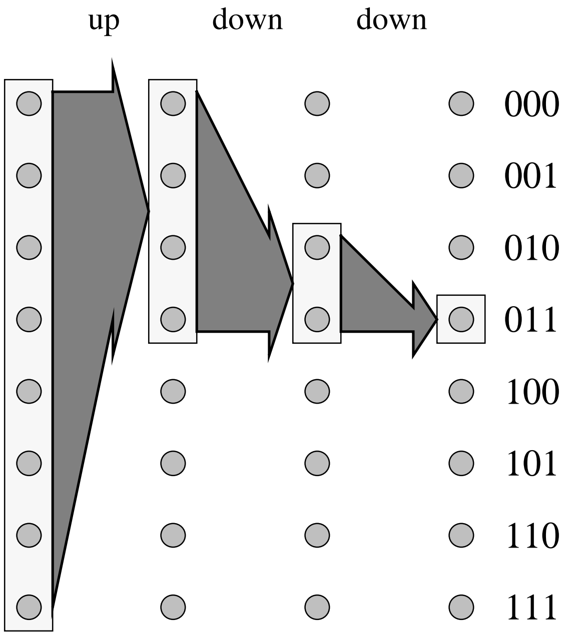

Figure 2: The logical path from any input to output 011.

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 delta network, each input and output are connected by a single logical (up-down) path through the blocks of the network. For example, Figure 2 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 general, however, each switch may have several up and down edges, say d of each, and each step of the logical path can be taken on any one of d edges.

1.3 Multibutterflies

A d-dilated butterfly can be thought of as d butterflies that are merged together by merging switches that have the same row and level numbers. In a recent paper, Upfal [17] proposed a more general way to merge butterfly

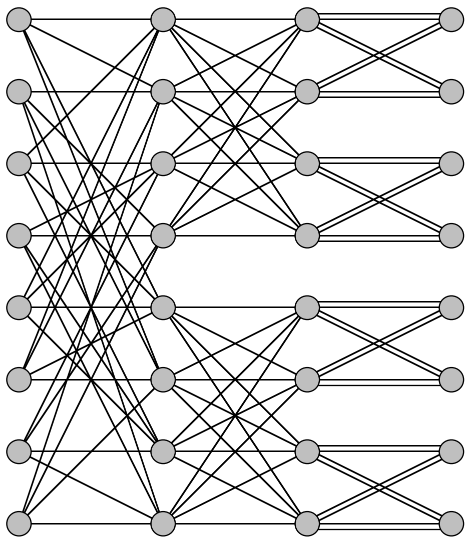

Figure 3: An 8-input 2-butterfly network.

networks. The idea is to permute the order of the switches within each block before merging the networks. Thus, given 2 N-input butterflies G_1 and G_2 and a collection of permutations $\Pi = \langle \pi_0, \pi_1, \ldots, \pi_{\log N} \rangle$ where $\pi_l : [0, \frac{N}{2^l} - 1] \rightarrow [0, \frac{N}{2^l} - 1]$, a 2-butterfly is formed by first permuting all of the switches in each block on level l according to π_l, for $0 \leq l \leq \log N$, and then merging switches with the same row and level numbers. The result, as shown in Figure 3, is an N-input graph with depth $\log N$ in which each switch has 4 inputs edges and 4 outputs edges. Of the 4 output edges at a switch, two are up edges and two are down edges (with one up and one down edge coming from each butterfly). Multibutterflies (i.e., d-butterflies) are composed from d butterflies in a similar fashion using $d-1$ sets of permutations, $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$, resulting in a depth $\log N$ network with $2d \times 2d$ switches.

1.4 Expansion

Dilated butterflies have remained the network of choice for many parallel machines. Recent work, however, suggests that this may be about to change. In fact, it now appears as though randomly-wired multibutterfly networks (i.e., multibutterfly networks where the permutations $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$ are chosen at random) are superior to dilated butterflies for many message routing applications. The crucial property that these networks possess is known as *expansion*. In particular, an M-input splitter is said to have (α, β)-*expansion* if any set of $k \leq \alpha M$ inputs is connected to at least βk up outputs and βk down outputs, where $\beta > 1$, $\alpha\beta < 1/2$, and α and β are fixed constants. Figure 4 shows a splitter with expansion (α, β).

Splitters with expansion $\beta > 1$ are known to exist for any $d \geq 3$, and they can be constructed deterministically in polynomial time [7, 15, 17], but randomized wirings typically provide the best possible expansion. In fact, the expansion, β, 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 [12, 17].) Furthermore, the constructions in Sections 3 through 5 require $\beta > d/2$, but, at present, there are no known deterministic algorithms for constructing splitters with this much expansion in polynomial time.

Splitters with expansion are good for routing since 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 classical networks such as the 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. Splitters with expansion can be constructed deterministically, but for $d \geq 3$, randomized wirings typically provide the best possible expansion.

1.5 History

Randomly-wired multibutterfly networks have been discovered several times. In 1974, Bassalygo and Pinsker [4] used random multibutterfly-like networks to construct the first nonblocking network of size $O(N \log N)$ and depth $O(\log N)$. In 1980, Fahlman [6] proposed a related randomly-wired network called the Hashnet. More recently, Upfal [17] coined the term multibutterfly, and provided a simple deterministic algorithm for routing any permutation of N messages in $O(\log N)$ steps on an N-input multibutterfly network. (In fact, Upfal's algorithm can be pipelined to route $\log N$ permutations in $O(\log N)$ steps.) Leighton and Maggs [13] then proved that a randomly-wired multibutterfly is an efficient routing network even when many of the switches are faulty. Later, Arora, Leighton, and Maggs [3] developed a

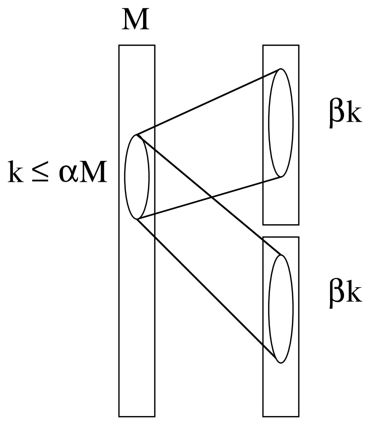

Figure 4: An M-input splitter with expansion property (α, β).

circuit-switching algorithm for the multi-Benes network and showed that it 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 delta network for processor to memory communications.

1.6 Outline

The remainder of this chapter is organized as follows. Section 2 describes Upfal's algorithm for packet switching on multibutterfly networks. Next, Section 3 presents a multibutterfly algorithm for circuit switching. Section 4 describes a strategy for tolerating faults. Finally, Section 5 sketches an algorithm for establishing connections in a randomly-wired nonblocking network.

2 Packet switching

In a one-to-one packet routing problem, each input sends a packet to a distinct output. The goal of the routing algorithm is to deliver the packets to their destinations as quickly as possible, subject to the constraint that at each time step, each edge can transmit at most one packet. There may also be restrictions on the number of packets that can be queued at any one switch.

In 1989, Upfal [17] proved that an N-input d-butterfly with expansion (α, β) can solve any one-to-one packet routing problem in $O(\log N)$ steps using a simple greedy algorithm. Moreover, he showed that by using pipelining, $O(\log N)$ problems can also be routed in $O(\log N)$ steps. The result is important because the only other known deterministic on-line linear-hardware $O(\log N)$-step packet routing algorithm [11] requires the use of the AKS sorting circuit [2] (which is more complicated and has larger constant factors).

2.1 The algorithm

Upfal's 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. 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 $L = \frac{1}{2\alpha}$ since then will we be guaranteed that at most $\frac{M}{2L} = \alpha M$ packets in any wave will ever pass through the up (or down) edges of any M-input splitter of the multibutterfly (for any M). 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.

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 connecting levels are colored in $2d$ colors so that each node is incident to one edge of each color. 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.

The following theorem summarizes the performance of Upfal's algorithm.

Theorem 2.1 ([13, 17]) *On an N-input multibutterfly with expansion (α, β), Upfal's algorithm routes P permutations in $O(P + \log N)$ steps.*

3 Circuit switching

In a one-to-one circuit-switching problem, each input wishes to establish a connection (path) to a distinct output. The connections must not intersect at any switch or edge. The goal of the circuit switching algorithm is to find the connections as quickly as possible.

In 1990, Arora, Leighton, and Maggs discovered an $O(\log N)$ bit-step algorithm for circuit switching on multibutterfly networks. The only previously known $O(\log N)$-bit-step algorithms for circuit switching relied on the AKS sorting circuit [2], or used randomness on the hypercube [1]. (Recently, Leighton and Plaxton have developed an $O(\log N)$-bit-step randomized sorting algorithm for the butterfly [14].)

3.1 Unique neighbors

The circuit-switching algorithm requires the splitters in the multibutterfly to have a special unique-neighbors property. An M-input splitter is said to have the (α, δ) *unique neighbor property* if in every subset X of $k \leq \alpha M$ inputs, there are δk nodes in X which have an up-output neighbor that is not adjacent to any other node in X, and there are δk nodes in X which have a down-output neighbor that is not adjacent to any other node in X. It is relatively easy to prove [3] that any any splitter with (α, β) expansion has the (α, δ) unique-neighbors property where $\delta = 2\beta/d - 1$, provided that $\beta > d/2$. Randomly-wired multibutterflies are known to have expansion (α, β) where $\beta > d/2$ [13, 17]. Explicit constructions of such splitters are not known, however.

3.2 The algorithm

In order for the algorithm to succeed, the number of paths passing through each M-input splitter must be at most αM. Thus, in an N-input network, we only make connections between the N/L inputs and outputs in rows that are multiples of L, where L is some fixed constant greater than $1/\alpha$.

There is a simple algorithm for extending paths from one level to the next in an M-input splitter with the (α, δ) unique-neighbors property. The basic idea is that those paths at switches with unique neighbors can be extended without worrying about blocking any of the other paths. Paths are extended by repeating steps of the following type. First, every unextended path sends out a proposal to his neighbors among the splitter outputs in the desired direction (up or down). Next, every output that receives precisely one proposal sends back its acceptance to that proposal. Finally, every unextended path that receives an acceptance advances to one of its accepting outputs. In each step, the fraction of unextended paths drops by a factor of $(1-\delta)$. Thus, after $O(\log M)$ phases, all of the paths are extended. By applying this algorithm one level at a time, it is possible to establish paths from the inputs to the outputs of an N-input multibutterfly with the (α, δ) unique-neighbors property in $O(\log^2 N)$ bit-steps.

A more sophisticated algorithm is needed to construct the paths in $O(\log N)$ bit-steps. Given a set of paths that need to be extended at an M-input splitter, the algorithm does not wait $O(\log M)$ time for every path to be extended before it begins the extension at the next level. Instead, it executes path extension steps until the number of unextended paths falls to some fraction, ρ, of its original value, where ρ is fixed constant that depends on d. Then the path extension process can start at the next level. The danger is that the paths left behind may find themselves blocked by the time they reach the next level. To ensure that this doesn't happen, stalled paths send out *place-holders* to all of their neighbors at the next level, and henceforth the neighbors with place-holders participate in path extension at the next level, as if they were paths. Of course, the neighbors holding place-holders must in general extend in both the upper and the lower output portions of the splitter, since they don't know yet which path will ultimately use them. Notice that a place-holder not only reserves a spot that may be used by a path at a future time, but also helps to chart out the path by continuing to extend ahead.

In order to prevent place-holders from multiplying too rapidly and clogging the system – since if the fraction of inputs of a splitter which are trying to extend rises above α, the path extension algorithm ceases to work – we need to ensure that as stalled paths get extended, they send *cancellation signals* to the placeholding nodes ahead of them to tell them they are not needed anymore. When a placeholding node gets cancellations from all the nodes who had requested it to hold their place, it ceases its attempts extend. It also sends cancellations to any nodes ahead of it that may be holding a place for it.

The $O(\log N)$-step algorithm alternates between two types of phases. First, paths extension steps are executed until the fraction of unextended paths in each splitter drops by a factor of ρ. In this phase, each path is restricted to extend forward by at most one level. We refer to the first wave of paths and placeholders to arrive at a level as the *wavefront*. The wavefront moves forward by one level during each phase. If a path or placeholder in the wavefront isn't extended, then at the end of the phase it sends placeholders to all of its neighbors. In the second phase, cancellations are passed through the network. They travel a distance of C, where C is some fixed constant that depends on ρ and d.

The performance of the circuit-switching algorithm is summarized in the theorem below.

Theorem 3.1 ([3]) *On an N-input multibutterfly with expansion (α, β), $\beta > d/2$, the algorithm solves any one-to-one circuit-switching problem in $O(\log N)$ bit steps.*

4 Fault tolerance

In 1989, Leighton and Maggs [13] showed that multibutterfly networks are highly fault tolerant. In particular, they proved that no matter how an adversary chooses k switches to fail, there will be at least $N - O(k)$ inputs and $N - O(k)$ outputs between which permutations can be routed in $O(\log N)$ steps. Note that this is the best that could be hoped for in general, since the adversary can choose to make $\Omega(k)$ inputs and $\Omega(k)$ outputs faulty. Thus, the multibutterfly is the first bounded-degree network known to be able to sustain large numbers of faults with minimal degradation in performance.

4.1 The strategy

The strategy for tolerating faults consists of two stages, erasure of outputs and fault-propagation.

Each splitter in the multibutterfly is examined in the erasure stage. If more than an ε fraction of the splitter inputs are faulty, where $\varepsilon = 2\alpha(\beta' - 1)$ and $\beta' = \beta - \lfloor \frac{d}{2} \rfloor$ then the splitter as well as all descendant switches and outputs are erased from the network. The erasure of an M-input splitter causes the removal of M multibutterfly outputs, and accounts for at least εM faults. Hence, at most $\frac{k}{\varepsilon} = \frac{kL}{\beta'-1} = O(k)$ multibutterfly outputs are removed by this process.

Next, working from level $\log N$ back to level 0, each switch is examined in the fault-propagation stage to see if at least half of its upper outputs lead to faulty switches that have not been erased, or if at least half of its lower outputs lead to faulty switches that have not been erased. If so, then the switch is declared faulty (but not erased). It is not difficult to prove [13] that at most $\frac{k}{\beta'-1}$ additional switches are declared faulty at each level by this process. Hence, at most $O(k)$ multibutterfly inputs will be faulty (declared or otherwise).

We now erase all the remaining faulty switches. This leaves a network with $N - O(k)$ inputs and $N - O(k)$ 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 Theorems 2.1 and 3.1 with β replaced by β' to show that the network can solve packet-switching and circuit-switching problems on the working inputs and outputs in $O(\log N)$ steps.

5 Nonblocking networks

In a *nonblocking network*, inputs are connected to outputs with node-disjoint paths, as they were in Section 3. The inputs, however, are not all required to make their requests for connections at the same time. Inputs may wait to make their requests, and may later break connections and request new ones. The main invariant obeyed by a nonblocking network is that any unused input–output pair can be connected by a path through unused switches, no matter what paths have previously been established. The 6-terminal graph shown in Figure 5 is an example of a nonblocking network. In particular, if Bob is talking to Alice and Ted is talking to Carol, then Pat can still call Vanna.

The existence of a bounded-degree strict-sense nonblocking network with size $O(N \log N)$ and depth $O(\log N)$ was first proved by Bassalygo and Pinsker [4] in 1974. Unfortunately, there has not been much progress on the problem of setting the switches so as to realize the connection paths since then. Until recently, no algorithm was known that could cope with

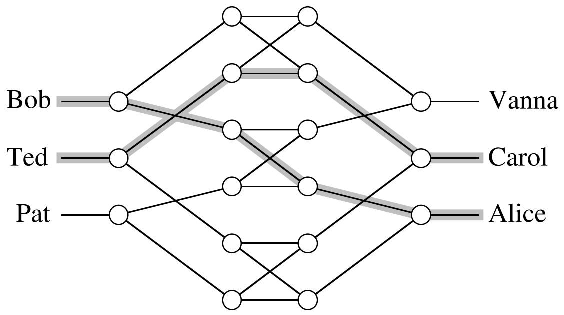

Figure 5: A nonblocking network with 3 inputs and 3 outputs.

simultaneous requests for connections in any $O(N \log N)$-size nonblocking network.

In 1990, Arora, Leighton, and Maggs discovered an $O(N \log N)$-switch nonblocking network for which each path connection can be made on-line in $O(\log N)$ bit-steps. The algorithms work even if many calls are made at once — every call still gets through in $O(\log N)$ bit-steps, no matter what calls were made previously and no matter what calls are currently active, provided that no two inputs try to access the same output at the same time.

5.1 Multi-Benes networks

The nonblocking network is called a *multi-Benes network*. A multi-Benes network is constructed by combining expanders and the Benes network in much the same way that expanders and butterflies are combined to form a multibutterfly. As shown in Figure 6, a multi-Benes network is essentially a reversed multibutterfly followed by a multibutterfly.

As in the circuit-switching algorithm, the network must be lightly loaded by some fixed constant factor L, where $L > 1/\alpha$. Since the other inputs and outputs are not used, the first and last $\lg L$ levels of the network can be removed, and the N/L inputs and outputs can each be connected directly to their L descendants and ancestors on levels $\lg L$ and $2 \lg N - \lg L$, respectively.

The basic idea is to treat the switches through which paths have already been established as if they were faulty and to apply the fault propagation techniques from Section 4 to the network. In particular, we define a node to be *busy* if there is a path currently routing through it, and we recursively

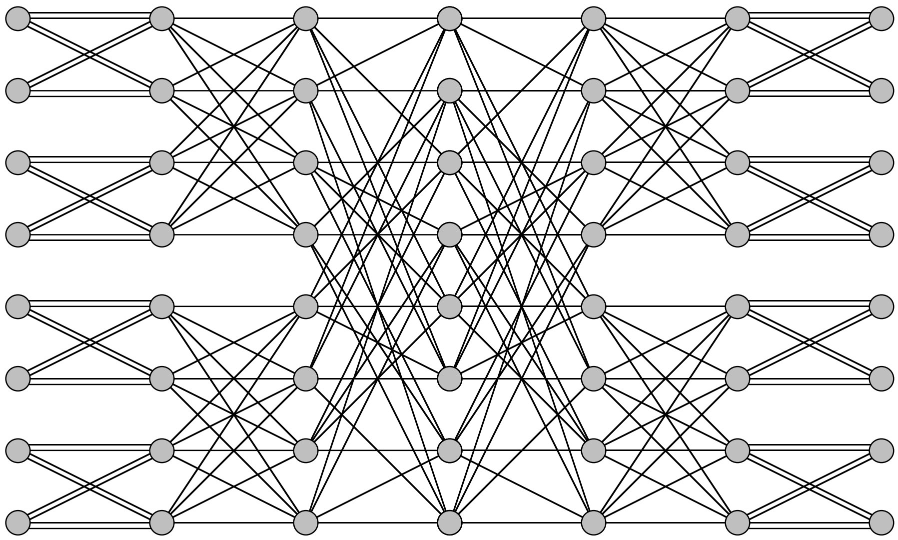

Figure 6: An 8-input 2-multi-Benes network.

define a node to be *blocked* according to the following rule. Working backwards from level $2\lg N - \lg L - 1$ to level $\lg N$, a switch is declared blocked if more than $2\beta - d - 1$ of its up (or down) neighbors on level $l+1$ are busy or blocked. From level $\lg N - 1$ to level $\lg L$, a switch is declared blocked if more than $4\beta - d - 2$ of its $2d$ neighbors on level $l+1$ are busy or blocked. A switch that is neither busy nor blocked is said to be *working*.

Two important properties can be proved about the network switches. First, for $\beta > 2d/3 + 2/3$ and $L > 1/2\alpha(3\beta - 2d - 2)$, at most a 2α fraction of the switches in any block are declared to be blocked. Thus, all of the unused inputs are working. As a consequence, no matter what paths have already been established, any unused input can reach any unused output. Second, for $\beta > d/2$, the network of working switches has a $(\alpha, 1/d)$ unique-neighbor property. As a consequence, the circuit-switching algorithm from Section 3 can be used to establish new paths, even if many requests for connections are made simultaneously.

References

[1] W. Aiello, T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 55–64, July 1990.

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

[3] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 384–389. IEEE Computer Society Press, October 1989.

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

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

[16] 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.

[17] 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.