Adaptive Routing for Fault Tolerance and Performance

Eric A. Brewer

February 4, 1993
Revised February 17, 1994

1 Introduction

This paper examines the use of adaptive routing in multiprocessors to improve fault tolerance and network performance. In particular, I examine adaptive routing for k-ary n-cubes and multibutterflies given the myriad of limitations imposed by current and near-future technology. The primary question is whether or not adaptive routing is effective given the pragmatics of current technology.[1]

The primary conclusions are that adaptive routing should be used in multiprocessors and that, *using nonminimal adaptive routing*, k-ary n-cubes achieve better fault tolerance and performance than multibutterflies. I also present several principles for designing adaptive routers to maximize their performance and fault tolerance.

Section 2 covers the array of background material required for later analysis and discussion, and Section 3 introduces multibutterflies. The meat begins in Section 4 with a detailed analysis of fault tolerance and congestion, and continues through Section 6. Section 5 examines the problems with multibutterflies and proposes a related network to address these problems. Section 6 covers the relative performance of deterministic and adaptive routing and shows how to maximize the performance of adaptive routing. Finally, Section 7 summarizes the criticisms of the three papers, and Section 8 presents the overall conclusions.

2 Background

This section covers a range of background material that will be needed throughout the paper. After some definitions, this section covers the semantic properties of adaptive routing, the four basic routing styles, the prevention of deadlock, and the issues involved with using long wires in a network.

2.1 Some Definitions

Deterministic Routing A routing algorithm is *deterministic* or *oblivious*, if the decision at each switch depends only on the destination. Deterministic routers provide exactly one path between any pair of processors, which is the key to their poor performance and fault tolerance.

[1] This paper is the result of MIT's area exam, in which students examine a broad question that is outside their primary area. The papers assigned for this exam were [LM92], [Dua91a], and [KS90].

Minimal For every pair of endpoints there is a shortest path through the network; there may be many paths of that length. A *minimal* routing algorithm always routes along a shortest path. In practice, all deterministic routers are minimal.

Nonminimal A routing algorithm is *nonminimal* if it may use longer paths. Since nonminimal routers use a superset of the paths of minimal routers, they are more fault tolerant. Typically nonminimal routers use longer paths only to avoid congestion or faults.

Fully Adaptive Routing algorithms that can route along all of the shortest paths are called *fully adaptive*. In practice, nonminimal routers are fully adaptive, while minimal routers may or may not be.

Channel A (physical) *channel* is an edge in a network topology and corresponds to set of wires.

Flit A *flit* is the smallest unit upon which the router performs flow control.

Head The *head* of a message is the first flit; it contains information on the destination that is used to route the message. (Sometimes the head consists of multiple flits.)

Tail The *tail* of a message is the last flit. For some kinds of routing, it is a distinguished pattern that releases resources as it goes.

2.2 Semantic Properties

A routing algorithm may be judged according to five semantic properties: freedom from deadlock, livelock, and starvation, fault tolerance, and preservation of message order.

Freedom from Deadlock A critical property of routing algorithms is freedom from deadlock. A set of messages is deadlocked when each message in the set requires a resource held by another message in the set. A proof that a routing algorithm is deadlock free generally consists of showing that resource dependencies can not be cyclic, so that given a set of blocked messages at least one does not depend on anything held by another message in the set. Such proofs require the *consumption assumption*, which states that target processors must pull arrived messages out of the network.

Freedom from Livelock Even if a set of messages is not blocked, the messages may not necessarily make forward progress. Livelock occurs when a set of messages continuously moves through the network without ever reaching their targets. Only nonminimal routers can experience livelock: minimal routers use only paths with a fixed finite length, so moving messages must eventually reach their targets.

Freedom for Starvation Starvation occurs when a processor is unable to inject a message for a very long time due to congestion. This typically occurs when a router has preferred inputs, that is, it always selects one input over another when both offer messages. The standard technique for preventing starvation is handling the inputs in round-robin order, typically by passing around an "ok to transmit" token among the inputs. Starvation is not as severe as deadlock or livelock, since the traffic will eventually drop off. For example, the J-Machine router [ND92] always chooses passing traffic over messages from the local processor. In one application that contains large data-exchange phases [Noa92], some processors are unable to inject a message until nearly all other processors have completed the phase. Although this long delay severely hurts performance (because the other processors must wait for it to catch up), the application does complete correctly.

Fault Tolerance Fault tolerance is a property of the network as a whole, but the routing algorithm is the key element. The key issue is whether or not the network ensures uncorrupted delivery of all messages to all reachable nodes. If not, a fault-tolerant system must be prepared for retransmission, which increases the software overhead of sending messages. For example, the J-Machine can inject a message with an overhead of only 11 cycles [NWD93], while support for retransmission (using sequence numbers) takes about 14 cycles [BCH+92] and increases message length. Section 4 discusses fault tolerance in detail.

Preservation of Message Order Message-order preservation is also a property of the network that is determined by the routing algorithm. A network preserves order if and only if two messages sent from A to B always arrive in the order sent. Deterministic routing algorithms preserve message order if they ensure delivery, since all messages travel along the same path. Adaptive routing algorithms generally do not provide message order since the second message may travel along a faster (less congested) path than the first. The cost of reordering messages in software is only slightly less than the cost for supporting retransmission. If the software already supports retransmission, then the additional overhead for reordering messages is typically zero [BCH+92].

2.3 Basic Routing Styles

All routing algorithms are based on one of four basic styles: packet-switched, circuit-switched, virtual cut-through, and wormhole. These definitions are based on Konstantinidou's Ph.D. thesis [Kon91] and on DeHon's Master's thesis [DeH93], although the analysis is largely my own.

2.3.1 Circuit-Switched Routing

In a circuit-switched network, a path is first established from sender to target and reserved for the current message. Any adaptive routing decisions are made during this phase. The message is then sent along the path with the tail releasing the reserved channels as it crosses them.

There are two approaches to establishing the path. The traditional approach [Gru89] is to determine the path completely before sending any of the data. If congestion is encountered, then the either the path is stalled until it can make progress or back-tracking is used to try a different route. The problem with this approach is that all of the reserved channels remain idle until the path is *completely* established and transmission begins.

The second approach [DeH93] optimistically assumes that a path can be established without blocking or back-tracking. Essentially, the path determination and transmission are pipelined: the head determines a path as it goes and the data follows immediately behind. There is little storage in each router, so if the head is unable to make progress the transmission must be aborted. This is obviously a waste of resources, so this approach depends on low contention, usually through a large number of paths. It should be noted that aborting transmission exacerbates congestion since it uses channels without making any progress; this reduces the achievable throughput.

A second problem with this approach in practice is that the channels must be held until an acknowledgment reaches the sender. Otherwise, the sender cannot tell if the message was aborted. For large networks and short messages, this implies that many channels will be idle waiting for the acknowledgment.

Although circuit switching wastes bandwidth, is does provide one advantage: it incurs little software overhead for fault tolerance and message-order preservation. This is because the sender always knows that target received the message; messages remain in order since there is no overlap in transmission.

2.3.2 Packet-Switched Routing

For packet-switched routing, also called *store-and-forward* routing, the entire message moves from node to node as a unit. The advantages are that each message uses exactly one channel at a time and that the contents can be verified (and fixed) on each hop. There are two serious disadvantages. First, a message remains in a node until all of the message has entered the node, even if the desired output channel is available. Second, there must be sufficient buffer space on the node for an entire message for every input. This requires significant area and in practice limits the size of messages to 32 to 64 flits [KS91].

2.3.3 Virtual Cut-Through Routing

Virtual Cut-Through routing addresses the unnecessary delays imposed by packet switching [KK79]. In particular, the head of a message never waits for the tail to reach the current node; it simply keeps moving until it encounters congestion or reaches the target. Although this greatly improves throughput and latency, the storage requirements remain the same as packet switching: there must be sufficient storage in each output buffer for the longest message.

A key property of virtual cut-through routers is that a blocked message only blocks an incoming message if that message requires the same output channel. Thus incoming messages may pass blocked messages, which increases achievable throughput.

2.3.4 Wormhole Routing with Virtual Channels

Wormhole routing [DS87] can be viewed as either a variation of virtual cut-through or a variation of circuit switching. It is similar to virtual cut-through except that each node contains a small amount of buffering, not enough for an entire message. This means that messages cover many channels simultaneously. Under zero contention, wormhole routing behaves identically to virtual cut-through.

Unlike virtual cut-through, a blocked message ties up many channels (all of the channels it occupies), which causes them to remain idle. The solution to this problem is *virtual channels* [Dal90b]. Each physical channel is multiplexed into several virtual channels, each with their own small buffer. The key point is that messages on different virtual channels can not block each other, even though they use the same wire. Thus, a blocked message only ties up one virtual channel — messages on other virtual channels may pass it, thus making use of the wire and increasing the achievable throughput. For an 8-stage butterfly, Dally shows that the achievable throughput improves from about 30% of capacity with one channel to nearly 70% with 8 virtual channels per physical channel [Dal90b].

The key advantage of wormhole routing over virtual cut-through is the reduction in required buffering. The buffering is fixed at a few flits per virtual channel, which simplifies the router. Simulations show that for a given amount of buffer space, using many virtual channels provides much better throughput than using one channel with a large buffer (as required by virtual cut-through routing) [Mai88].

Wormhole routing is similar to pipelined circuit-switching in that the head establishes the path and the tail releases resources. The key difference is that under congestion a message is blocked rather than aborted. Additionally, any acknowledgment uses a separate path and message, so that channels are not idle in the interim.

Wormhole routing has become the preferred style because of its low latency, high achievable bandwidth and low buffering requirements. The extensive use of blocking complicates proofs of freedom from deadlock, but several deadlock-free adaptive wormhole routing algorithms exist [Dua91a, LH91, DA92, BGP92, YJ89, GN92, CK92a].

A major criticism of the Chaos router is that it uses virtual cut-through routing [KS90, KS91, Kon92]. For the same buffering space, it would achieve higher throughput using wormhole routing and could avoid a strict limit on message size. The only drawback would be having to ensure freedom from deadlock, which is clearly possible. The next section examines such proofs.

2.4 Freedom from Deadlock

Different routing styles require different techniques to generate deadlock-free routers. This section examines these techniques. In all cases, the basic idea is to prove that there can not be cyclic resource dependencies; in particular, there can not be a cycle of blocked messages.

All of the proofs for freedom from deadlock require the consumption assumption: messages that reach their destination must be pulled in by the processor. Given the finite buffer space in real processors this may be nontrivial to ensure, but it is achieved in practice [L$^+$92, D$^+$89].

An easy way to ensure freedom from deadlock is to pick a topology that has no cycles. For example, multistage networks have the property that messages always advance from stage to stage: there are no backward edges and thus no cycles. The fat-tree of the CM-5 employs this strategy [Lei85, L$^+$92]. For networks that intrinsically contain cycles, such as k-ary n-cubes, more complicated techniques are required.

Circuit-switched networks that abort messages, rather than block them, are deadlock free simply because there are never any resource dependencies. However, blocking circuit-switched routing must be treated as wormhole routing in terms of freedom from deadlock.

Packet-switched networks and virtual cut-through networks use a *structured buffer pool* to prevent deadlock. Essentially, the buffers are partitioned into classes that obey a partial order. The assignment of buffers to messages is restricted to prevent cyclic dependencies [Gun81, MS80, Tou80, TU79, Gel81]. Such approaches do not apply to wormhole routing because they assume that the incoming channel is free once the message is buffered [DS87].

2.4.1 Wormhole Routing

The key to proofs of freedom from deadlock for wormhole routing is to show that the dependencies between channels allowed by the routing algorithm form an acyclic graph. There is a *direct dependency* from channel i to channel j if and only if j immediately follows i in some path allowed by the routing algorithm. A key point is that adjacent edges in the topology may not be dependent, since the routing algorithm may prohibit that particular turn.

The *channel dependency graph* has one node for each channel and an edge from node i to node j if and only if there is a direct dependency from channel i to channel j. One way to prove freedom from deadlock is to show that this graph is acyclic. This approach was used for the original deadlock-free wormhole routing algorithm proposed by Dally [DS87] and is the essence of Theorem 1 in Duato's methodology for the design deadlock-free routing algorithms [Dua91a, Dua91b]. Because the channel dependency graph reflects the routing algorithm, topologies with cycles, such as k-ary n-cubes, can have acyclic graphs.

Restricting the routing algorithm to eliminate cycles may leave the topology effectively disconnected: not all pairs of nodes may be able to communicate given the routing restrictions. Virtual channels are introduced to reconnect the topology (without adding a cycle to the channel dependency graph) [DS87, Dua91a].

Duato's second theorem shows that routing algorithms can be deadlock free even if there are cycles in the channel dependency graph. The key invariant is that a fully connected subset of the channels must have an acyclic dependency graph. (This graph is called the *extended channel dependency graph.*) The key idea is that anytime there is a cycle of blocked messages, the one with the lowest channel number can simply follow the acyclic subset (which must be present since the subset is fully connected). In essence, the routing reverts to a deadlock-free subset whenever a cycle if formed, thus breaking the deadlock. In practice this means that a channel can not block on a channel with a higher number[2], although it can use the channel when there is no congestion.[3]

Using Duato's methodology leads to algorithms that are usually adaptive, but revert to deterministic algorithms if needed to avoid deadlock. Although in theory the restricted routing algorithm could be adaptive, all of Duato's examples are either deterministic are have deterministic subsets themselves. Nearly all of the adaptive wormhole routing algorithms have this property [DA92, CK92a, BGP92]. One exception is the "turn model" for adaptive routing developed by Glass and Ni [GN92], which restricts turning to eliminate cycles in the channel dependency graph.

2.5 Wires

Wiring dominates the cost, power consumption, and area required for large-scale multiprocessors. For two- and three-dimensional meshes and tori, the wires are uniformly short. In this case, the cycle-time (for synchronous systems) is limited by the sum of the time off the node, across the wire, and on to the target node.

With the exception of two- and three-dimensional meshes and tori, nearly all network topologies require long wires, including multibutterflies [Upf89], express cubes [Dal91] and fat-trees [Lei85].

2.5.1 Long Wires

For long wires, the latency across the wire is proportional to the length of the wire. This implies that high-performance systems must pipeline bits onto long wires; limiting the cycle time to accommodate the transit time of the longest wire severely reduces performance.

[2]Although it can block for a *finite* amount of time [DA92].

[3]To avoid deadlock, a message that takes such a channel must be able to advance all the way to the head of the input queue. If it stops in the middle of the queue, it cannot exit through the deadlock-free subset, since only messages at the head can make routing decisions.

Pipelining bits converts long-latency, low-throughput wires to long-latency, high-throughput wires. For example, a 200 MHz signal has a period of 5 ns, which corresponds to about 75 cm given that the speed of light through a wire is about 15 cm/ns. Thus, a 3-meter wire is $\frac{3}{0.75} = 4$ cycles long. The throughput is the full 200 Mb/s, but the latency is 4 "short" cycles rather than one "long" cycle. In practice, pipelining bits is complicated but achievable [L+92].

Although the latency of long wires is several clock cycles, it is clear that using long wires is faster than making several hops along short wires to cross the same distance. Each hop takes at least one cycle and will advance the data only a small fraction of the length of one cycle along a wire.

Long wires also suffer from *reflection noise*, in which a signal interferes with its reflection, and *crosstalk*, in which a group of wires induce noise in each other [DLD93]. Fortunately, the long wires in networks are point-to-point connections. Thus, designers can use impedance matching to eliminate reflections, and use shielded coax cable to prevent crosstalk.

Another problem with long wires is power dissipation. A primary tool for reducing power requirements is to use smaller voltage swings. Moving from 5-volt swings to 1-volt swings leads to a factor of 25 reduction in power usage [Kni89]. Recent work [DLD93] provides 250-millivolt swings for inter-chip connections.

Synchronization between the target and the incoming data is also non-trivial for large-scale machines. The conceptually simplest approach is to ensure that all nodes reference the same global clock and the clock skew is a small fraction of the clock period. In this case, which is common today [TMC92, D+89], designers must ensure that all long wires are an integer number of cycles in length.

Alternatively, the receiver can *detect* the phase and clock rate of the incoming data and adjust accordingly [Mes90, DLD93]. This approach, called *plesiochronous interconnect*, uses a small amount of buffering at the receiver to handle slight differences in clock rate, and thus introduces some additional latency. Although somewhat more complicated, this solution is much more robust and allows wires to be any length.

Although these are challenging engineering problems, I will treat them as solved for the remainder of this paper. Two key conclusions are that long wires are required for most topologies and that long wires are not simple or free in any sense.

Finally, it should be noted that optical interconnect is a reasonable alternative for long wires. Optical fiber has extremely high bandwidth, no crosstalk or reflections, low signal attenuation, and is lighter and smaller than wire [GLKA89, Hsi90]. The speed of light is also about 25% faster in fiber than in copper wire, although more expensive kinds of wire, such as teflon-coated copper, are even faster [L+92]. The primary limitation is the low switching rate of current technology, which for the low power typical in VLSI chips is about 30 ns or about 15 cycles at 500 MHz [LHM+89]. This is essentially just extra latency, since many channels can use the same fiber to achieve bandwidth far greater than that using wires (for a given total cross-sectional area). The low attenuation makes fiber increasingly attractive as the distance increases; the crossover point in terms of power is a few meters (for the slow switching speeds given above). Also, the power requirements for fiber are constant relative to frequency, while for wire the attenuation increases with the square root of the frequency. Longer term, free-space and holographic optical networks are a possibility; Paturi et al. examine optical multibutterflies [PLF+91].

3 Multibutterflies

Multibutterflies are based on butterfly networks, which are multistage networks in which there are $\Theta(logP)$ stages for P processors. In general, a k-ary n-fly, which uses switches with k inputs and k outputs, connects k^n inputs to k^n outputs using n stages of k^{n-1} switches. The number of stages is equivalently viewed as $log_k P$ for P processors, since $P = k^n$.

Each switch routes to one of k outputs depending on the next $log_2 k$ bits of the destination address. For example, in a 2-ary n-fly each router routes up or down depending on the next bit, while in a 4-ary n-fly each router selects one of four outputs depending on the next 2 bits of the address.[4]

There is exactly one choice for each bit pattern, so there is exactly one path from one processor to another. Multibutterflies [Upf89, LM92] address this problem by providing multiple channels for each direction, so that although the direction is uniquely determined by the address, the exact channel is not. A 256-processor multibutterfly is shown in Figure 1.

The group of switches in one direction is called a *splitter*. In Figure 1, the first stage of 64 switches forms one splitter, and the second stage consists of 4 splitters, each with 16 switches. Each stage has k times as many splitters as the previous stage, each with $\frac{1}{k}$ times as many switches. Thus, the third stage consists of 16 splitters with 4 switches each.

The *dilation* of a multibutterfly is number of channels per direction; a standard butterfly has a dilation of one, while proposed multibutterflies have a dilation of two [DeH93]. It is also possible to have non-integer dilations. For example, 4 inputs might go to 6 outputs, which provides less congestion and better fault tolerance than a butterfly without doubling the number of wires.[5]

The last stage in Figure 1 consists of 128 dilation-1 switches instead of 64 dilation-2 switches. Without this modification, there would be one switch per splitter in this stage, which limits fault tolerance. Using 128 smaller switches, which requires the same total number of chips and wires, the two paths to an output always go through different chips [DeH93, CK92b].

The key property of multibutterflies is that they provide *expansion* [LM92]. An M-input splitter has $(\alpha - \beta)$ expansion if every set of $k \leq \alpha M$ inputs is connected to at least βk channels in each direction, where $0 < \alpha < 1$ and $\beta > 1$. A network has $(\alpha - \beta)$ expansion if all of its stages do. Expansion ensures fault tolerance because βk outputs must be faulty to prevent k inputs from routing. Similarly, many channels must be congested to cause congestion at the inputs.

3.1 Routing in Multibutterflies

The primary routing algorithm in the Leighton-Maggs paper [LM92] is for *permutation routing*, which routes a set of messages, one per input, in $\Theta(logP)$ global steps. In MIMD machines, processors do not work in global steps and messages are injected continuously. Thus, permutation routing is not realistic.

However, continuous routing works fine on multibutterflies, and all four basic routing styles could be used. Current research examines (continuous) circuit-switched routing on multibutterflies [DeH93, CED92]. Other work covers virtual cut-through routing [LLM90,

[4]Note that the address has length $log_2 P = log_2 k^n = (n)(log_2 k)$. Thus, after n stages all of the bits have been used exactly once.

[5]The existence of 4-to-6 switches implies that there must be *concentrators* in later stages, such as 6-to-4 switches. Otherwise the network would have more outputs than inputs.

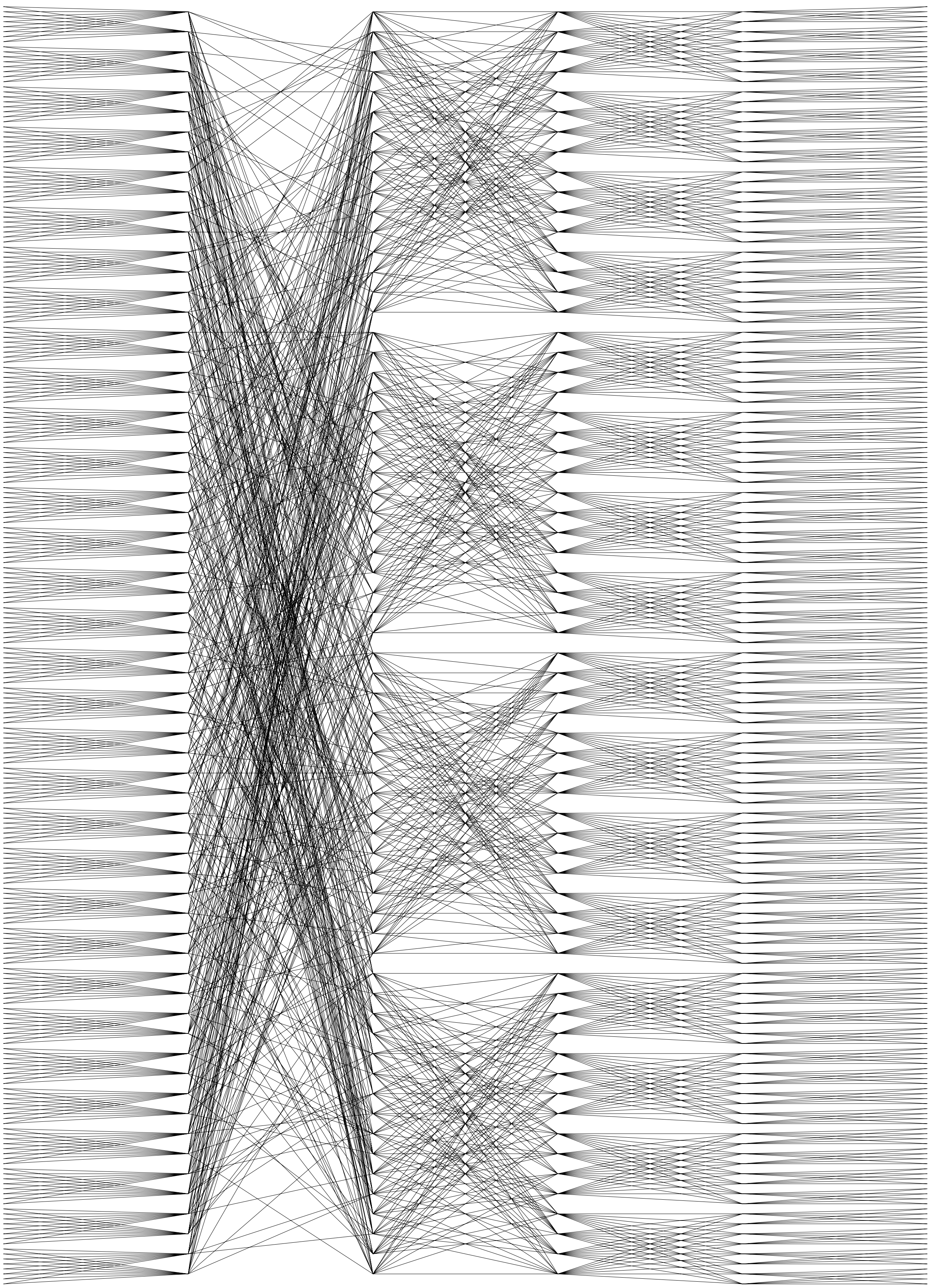

Figure 1: A 256-processor randomly-wired multibutterfly with radix-4 dilation-2 switches. The last stage consists of 128 radix-4 dilation-1 switches so that the two paths to an output do not use the same router chip. This schematic is consistent with the networks proposed by the MIT Transit group [DeH93, CED92, CK92b], and uses the wiring proposed by Leighton and Maggs [LM92].

KU91]. Since multibutterflies are acyclic, the use of wormhole routing is straightforward and deadlock free. However, several virtual channels should be used to reduce blocking and improve throughput [Dal90b]; 4 to 8 virtual channels per physical channel should be sufficient.

Routing in multibutterflies is minimal adaptive by definition, since all paths have the same length. Note that a deterministic algorithm could use only one output and would turn a multibutterfly into a traditional butterfly (with no fault tolerance). Thus, *adaptive routing on multibutterflies is practical only if multibutterflies are practical.* After investigating fault tolerance in the next section, Section 5 examines the practicality of multibutterflies and addresses some their shortcomings.

4 Fault Tolerance

In all topologies, the key to fault tolerance is to provide many paths between any pair of nodes. Deterministic routing algorithms provide no fault tolerance because there is exactly one path between any pair of nodes. In general, the more paths an adaptive routing algorithm can exploit the better its performance and fault tolerance.

For a given number of faults, the *yield* of a system is the probability that it will work 100% correctly, and the *harvest rate* is the expected percentage of the system that remains usable. For example, if we remove wires from a network, the yield is the probability that it remains 100% connected (end to end), and the harvest rate is the expected percentage of nodes that will still be able to communicate.

4.1 Expected Number of Faults

In the past year, the J-Machine has averaged roughly 256 nodes up and running, which amounts to 1024 chips and 768 channels. I will count nodes as 4 components, routers as 1 component, and channels as one-tenth of a component, so a 256-node J-Machine counts as 1100 components.[6] Excluding infant mortality, the J-Machine has had one fault [Noa]. Similarly, in 1989, 9000 Tandem systems with a total of 25,500 processors experienced a total of 157 hardware faults, or about 6 faults per 1000 processors, each of which has many components [Gra90].[7] Thus, assume that the failure rate for multiprocessors is 4 faults per year per 1100 components.[8]

Given k faults per year, how many simultaneous faults can we expect? Assume that the mean time to repair is less than a month and that faults are simultaneous if they occur in

[6] The 4:1 ratio for nodes to routers is consistent with the J-Machine and the Transit Project. The relative weight for channels is a rather uninformed estimate that implies that 10 channels are equally fault tolerant as one router. I was unable to find any evaluation of the actual relative fault tolerance. Fortunately, the component counts are used only for order-of-magnitude fault estimates, so the these ratios are adequate.

[7] This number includes some infant mortality.

[8] If the failure rate for the J-Machine actually is ≥ 4 faults per year, then the probability that only one occurs is 9.2%, so we are roughly 90% confident that the actual fault rate is < 4. This estimate is obtained by viewing the number of faults as a Poisson random variable with parameter λ. The probability density function for observing x faults is $f(x, \lambda) = \frac{e^{-\lambda}\lambda^x}{x!}$, so the probability density function for observing one fault is $e^{-\lambda}\lambda$. Thus, the conditional probability of observing one fault given that $\lambda \geq 4$ is:

$$\frac{\int_4^\infty e^{-\lambda}\lambda\, d\lambda}{\int_0^\infty e^{-\lambda}\lambda\, d\lambda} = \frac{e^{-\lambda}(-\lambda-1)\big|_4^\infty}{e^{-\lambda}(-\lambda-1)\big|_0^\infty} = \frac{0\ -\ e^{-4}(-5)}{0\ -\ (1)(-1)} = \frac{5e^{-4}}{1} = .0916 = 9.2\%.$$

the same month. If we view faults per month as a Poisson process with $\lambda = \frac{k}{12}$, we can bound the number of simultaneous faults. Poisson distribution tables tell us the probability of at most t faults in one month given that $\lambda = \frac{k}{12}$; call this value p. The probability that all 12 months have at most t faults is p^{12}.

For example, with 4 faults per year, $\lambda = 0.33$, the probability of at most one fault in one month is about 95.5%, and the probability that all 12 months have at most one fault is about 58%. The probability of at most two faults in one month is 99.5% and the probability that all 12 months have at most two faults is 94.4%. Thus, we are about 94% confident that there will be at most two simultaneous faults (given a mean time to repair on the order of a few weeks).

This implies that a 1024-node multibutterfly, which has 5 stages of 256 routers and 10240 channels, has about $1024 \cdot 4 + 5 \cdot 256 + \frac{10240}{10} \approx 6400$ components, and thus can expect at most 24 faults per year. The probability of at most 6 simultaneous faults is 94%. A 1024-node mesh, which has 1984 channels and thus about 4300 components, can expect 16 faults per year and at most 5 simultaneous faults (97% confidence). A 64K-node three-dimensional torus, which has 196,000 channels and thus about 282,000 components, might suffer from 1030 faults per year and at most 110 simultaneous faults (95% confidence). (These are conservative upper bounds.)

4.2 Fault Tolerance Comparison

This section compares the yield and harvest rate of meshes and butterflies under minimal and nonminimal routing. A fault consists of removing an edge from the topology. To represent bidirectional k-ary n-cubes I examine a mesh, since it has fewer paths and thus less fault tolerance than higher dimensional k-ary n-cubes. To facilitate comparison, I use topologies with roughly the same number of edges: I compare a 16x16 mesh, which has 480 edges, to a 64-processor dilation-2 multibutterfly, which has 512 edges (about 7% more edges).

For bidirectional edges, it is possible that only one direction may be affected by a fault. Thus, I will distinguish *edge faults*, which eliminate both directions, from *one-way edge faults*, which eliminate only one of two directions. For multibutterflies, which have strictly unidirectional edges, one-way edge faults are not applicable. It should be noted that two-way edge faults are more realistic and thus are used for most comparisons.[9]

4.2.1 Yield

Table 1 shows the expected number of edge faults until the network becomes disconnected. A network is *disconnected* if there exists an ordered pair of processors (A, B) such that A can not send messages to B. The numbers in Table 1 were generated by randomly removing edges until the network became disconnected. The procedure was repeated 100 times to get the expected number of faults and the 95% confidence interval for the mean.

Deterministic routing cannot tolerate any faults since every edge is used in at least one path and there are never any alternative paths. More surprisingly, *minimal adaptive routing does no better.* This is because every edge is the *only* path between its endpoints, thus that pair can no longer communicate.[10] Thus, any edge fault disconnects the network under

[9]This is because one-way communication requires that information move in both directions; there are relatively few places in which a fault would eliminate communication in only one direction.

[10]For unidirectional networks only one direction will be cut off, but the network is disconnected nonetheless.

Topology	Edges	Routing	Expected Faults	Percent
Any k-ary n-cube		Deterministic	1	0
Any k-ary n-cube		Minimal	1	0
16x16 mesh	480	Nonminimal	79 [73, 85]	16.5
32x32 mesh	1984	Nonminimal	264 [247, 281]	13.3
64-node Multibutterfly	512	(Minimal)	27 [25, 29]	5.3
Multibutterfly, double I/O	768	(Minimal)	120 [116, 124]	15.6
			One-Way Faults	
16x16 mesh	480	Nonminimal	132 [123, 141]	27.5
32x32 mesh	1984	Nonminimal	431 [401, 461]	21.7

Table 1: Yield Comparison for edge faults. The intervals in brackets are the 95% confidence intervals; they are based on 100 samples and are rounded outward from the mean.

minimal routing.

Nonminimal routing provides significant fault tolerance. From Section 4.1 we know that a 16x16 mesh can expect at most 2 faults at one time, while a 32x32 mesh can expect at most 5 faults. Thus, with nonminimal routing the network remains 100% connected with very high probability (for edge faults).

The normal multibutterfly is far less fault tolerant than a 16x16 mesh with nonminimal routing, even though the mesh has fewer edges. Doubling the number of channels for the input and output stages of the multibutterfly nearly triples the achievable fault percentage. Thus, the high fault tolerance reported by Leighton and Maggs [LM92] is due far more to the extra I/O connections than to properties of the multibutterfly itself (for machines of this scale). Section 4.2.3 examines the importance of the endpoints in more detail.

4.2.2 Harvest Rate

To estimate the harvest rate, I computed the percent connectivity after removing a given number of edges, using six samples for each point. Figure 2 shows harvest rate for four cases: minimal routing on a 16x16 mesh, nonminimal routing on a 16x16 mesh, and routing on a 64-node multibutterfly with and without double I/O connections. Figure 3 compares minimal and nonminimal routing on a mesh assuming one-way edge faults; the multibutterfly graphs are copied from Figure 2 for comparison.

As expected, nonminimal routing is more robust than minimal routing. More importantly, nonminimal adaptive routing on a mesh *provides better fault tolerance than a multibutterfly* (for a given number of total edges). This is true despite the fact that the mesh is the least fault tolerant k-ary n-cube, and has 7% fewer edges than the multibutterfly.

Interestingly, the multibutterfly provides better fault tolerance when the number of faults exceeds about 45% of the edges. However, this is irrelevant in practice since the upper bound on expected simultaneous faults is an order of magnitude less than this (for

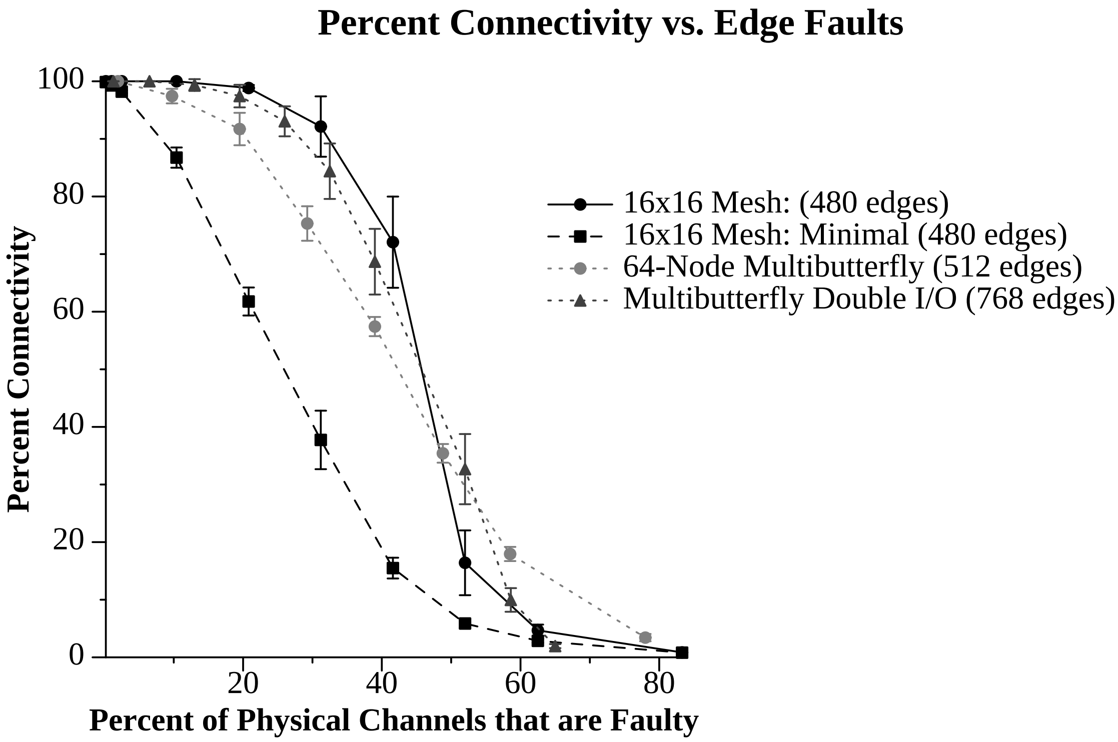

Figure 2: Harvest rate for edge faults. Values are the mean of six samples; the vertical bars delimit the 95% confidence interval.

---

machines of this scale). The high fault tolerance may imply less congestion; this issue is examined in Section 4.3.

With nonminimal routing the mesh stays 100% connected with more than 50 faults, which is far more than the expected number of two at one time. Minimal routing is better than 99% connected with two faults. The regular multibutterfly remains fully connected well above 10 faults, which exceeds the upper bound on expected faults of about two.

4.2.3 Endpoints are the Problem

Both the yield and the harvest rate for multibutterflies improved dramatically when the number of input and output channels was doubled. This implies that the number of connections to a node is the single most important factor in the fault tolerance of the network. This conclusion is consistent with the expectations of multibutterfly researchers [DeH93, LM92, CED92]. Figure 4 shows the size of the smaller partition when a 16x16 mesh first becomes disconnected. Over 60% of the time the size of the smaller partition is 1 node. In general, after a few hops the number of edges on the fringe of reachable nodes is vastly greater than the number of edges for the first or last hop. Nonminimal routing performs well primarily because it can use all possible edges for the first and last hop.

Thus, a key principle for designing adaptive routers is:

Design Principle 1: Endpoints are the key to fault tolerance.

This implies that a three-dimensional mesh is far more fault tolerant than a two-dimensional

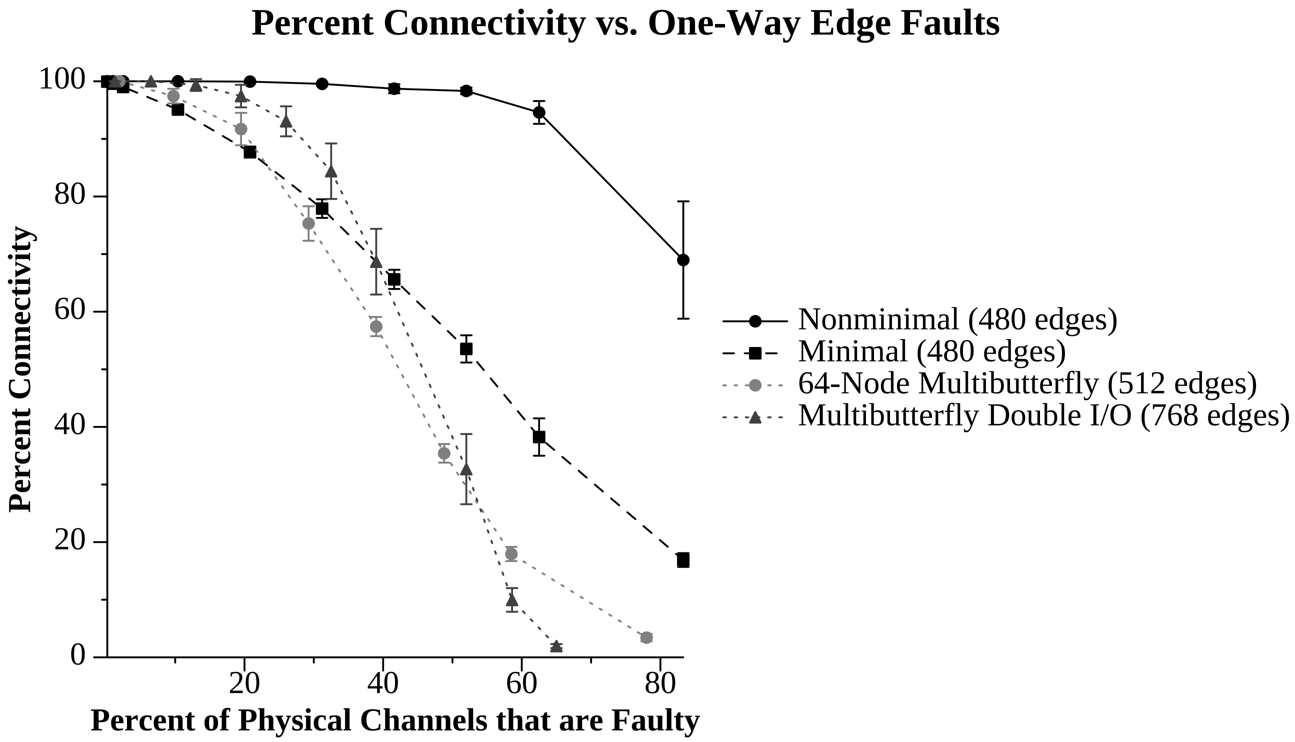

Figure 3: Harvest rate for one-way edge faults. Values are the mean of six samples; the vertical bars delimit the 95% confidence interval.

mesh, and that tori are better than meshes, since all nodes have the full number of connections.

4.3 Congestion

There is a natural connection between fault tolerance and congestion. A blocked channel looks essentially the same to a router as a faulty channel. Thus, a network's ability to handle congestion is linked with its fault tolerance. The percent connectivity can equivalently be viewed as the "percent of pairs that have a zero congestion path given a particular number of congested channels". Under this view it is clear that the mesh provides better *congestion tolerance* as well as better fault tolerance.

Although one-way edge faults are relatively unlikely, one-way congestion is quite likely. Thus, the data for fault tolerance for one-way edge faults is a good predictor for a network's ability to handle congestion. The results in Figure 3 reveal a huge advantage for nonminimal routing in k-ary n-cubes over both minimal routing and multibutterflies. Unlike faults, the expected percentage of congested channels may be quite high; thus the dramatic advantage of nonminimal routing on a mesh is important in practice.

This data also confirms the importance of bidirectional edges. With unidirectional edges, one-way edge faults are identical to two-way edge faults and the k-ary n-cubes would actually do worse than multibutterflies when more than 45% of the channels are congested. This is shown in Figure 2; the multibutterfly graphs are identical in the two figures. This leads to the second design principle:

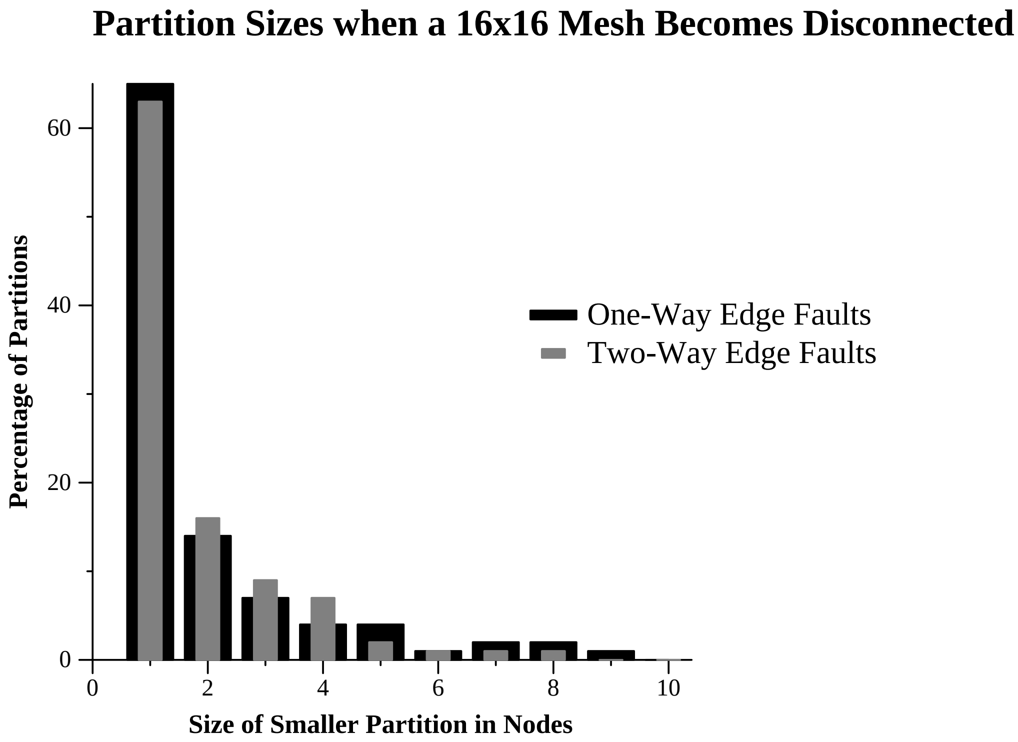

Figure 4: Distribution of partition sizes when a 16x16 mesh becomes disconnected.

Design Principle 2: Edges should be bidirectional.

The multibutterfly is clearly limited by the fact that it only uses wires in one direction. More generally, bidirectional channels double the number of paths in the network.

4.4 Conclusions

Although both k-ary n-cubes with nonminimal routing and multibutterflies provide sufficient fault tolerance to keep the network fully connected, k-ary n-cubes provide better congestion tolerance. Thus, although the multibutterfly has far superior fault tolerance and freedom from congestion than a butterfly [LM92, KU91, LLM90], it is not as good as a k-ary n-cube that uses nonminimal adaptive routing.

Nonminimal routing is clearly superior to minimal routing, both in yield and harvest rate. Since nonminimal paths have longer latency, routing algorithms should be "mostly" minimal; that is, routers should only misroute when it is required by congestion or faults. The third design principle is thus:

Design Principle 3: Routing should be nonminimal under duress.

Unfortunately, not all nonminimal routing algorithms can actually achieve the performance and fault tolerance implied by the above results, although the Chaos router can. In particular, the wormhole routing algorithms that revert to deterministic routing to avoid deadlock will achieve lower performance and fault tolerance [Dua91a, DA92, CK92a, BGP92].[11] When the routing switches to the deadlock-free subset, as discussed in Section 2.4, there may not be a path from its current position, even if the network is fully connected under full nonminimal routing. Thus, *even with one fault these algorithms can not ensure delivery.*[12]

Fortunately, the probability of being unable to deliver the message decreases greatly as the number of virtual channels is increased, since a reversion to the subset becomes very unlikely. However, adding virtual channels requires more area, more control logic, and perhaps a longer message header [CK92a, DA92].

One possible solution is to design the subset based on the turn model proposed by Glass and Ni [GN92]; this would make the subset partially adaptive, nonminimal, and still deadlock free. This proposal technically fits within Duato's methodology, although his examples and the literature in general always use a deterministic subset.

The primary reason the Chaos router can achieve the full performance of nonminimal routing is that it never reverts to a subset. It depends on randomization to avoid livelock with very high probability, rather than depending on the fact that the algorithm becomes deterministic after a bounded number of routing decisions.

5 Making Multibutterflies more Practical

From Section 4 we know that multibutterflies are neither more fault tolerant nor less affected by congestion than k-ary n-cubes under nonminimal routing for medium-scale machines. Furthermore, for medium-scale machines, those with say 64 to 1024 nodes, the multibutterflies have far more components; a 256-node multibutterfly has $64 \cdot 4 = 256$ components that are not needed for a 256-node k-ary n-cube.

Asymptotically, the multibutterfly is more fault tolerant than k-ary n-cubes. Leighton and Maggs [LM92] show that the multibutterfly can withstand a constant probability of faults and still route all messages with very high probability. Similar bounds for bounded-degree k-ary n-cubes exist but they require more complicated routing and relatively low probabilities [Rag89, HLN89]. Unfortunately, although multibutterflies may become superior for a large enough machine, it is not practical to build large-scale multibutterflies.

One severe problem is revealed in 256-node multibutterfly in Figure 1: the first stage has a huge number of randomly connected long wires. For machines with 256 nodes or less, Knight presents some creative three-dimensional packaging for dealing with these wires [Kni89].[13] For a 64K-node machine, each of the 128K channels in the first stage would be a long cable with distinct endpoints (i.e., different boards), as would the 128K channels in each of stages 2 through 4. (There are 8 stages in a 64K-node multibutterfly.) This is not realistic to build. For comparison, a fat tree might have similar number of long channels, but there are a very few endpoints, so huge groups of channels can be routed together. For example, the 32K channels to the root of a 64K-node fat tree can be grouped together; they will go to a large number of boards, but the boards will be located together. The assignment of channels to boards is pretty arbitrary and thus low labor; in the multibutterfly the

[11] Furthermore, the only nonminimal algorithm in this class [DA92] depends on the deterministic subset to ensure freedom from livelock.

[12] Planar adaptive routing [CK92a] is particularly susceptible to this problem, since it routes in two-dimensions at a time and becomes deterministic (until it switches planes) once the secondary dimension has reached the correct value.

[13] This limit of 256 nodes is Knight's estimate.

assignments are important.[14]

The multibutterfly also exhibits zero locality: all messages must traverse all stages. This implies higher latency and lower achievable bandwidth than a network with locality and the same number of channels.

Assuming $100\,\mathrm{cm}^3$ per node and $25\,\mathrm{cm}^3$ per router, we get a total volume of $65,536 \cdot 100 + 16,384 \cdot 8 \cdot 25 = 9.8 \times 10^6\,cm^3$ = 9.8 cubic meters, not counting wires (yet).[15] Starting with a 8 cubic meters, 2 meters on a side, we can place each of the four first-stage splitters in a 1 m by 1 m by 2 m volume, which leads to length of about 2 meters for the first stage channels. Making two more 4-way splits leads to 2-meter wires for the second stage and 1-meter wires for the third stage. We now have 64 cubes $\frac{1}{2}$ meter on a side and 5 more stages to place. If s is the length of one side of the starting cube, then the total length is:

$$(s + s + \tfrac{s}{2})(1 + \tfrac{1}{4} + \tfrac{1}{16} + \cdots) = \tfrac{5s}{2}(1.33) = 3.33s = 3.33\sqrt[3]{Volume}$$

Thus, the input-to-output wire length is about 7.2 meters if we could fit everything into 10 cubic meters. Since there are $2 \cdot 65536$ wires per stage, the volume for just the wires is $2 \cdot 65536 \cdot 7.2 \cdot \rho \cdot w$ for a cross-sectional area of ρ square meters per wire and w wires per channel. With ρ set to one square millimeter, which is reasonable for coax, and $w = 4$, the wire volume is about 4 cubic meters. Thus, if we take our original volume estimate of 10 cubic meters plus 4 cubic meters for wire and recompute the wire length, we get about 8 meters (where this process converges). This is about 30 ns of latency for teflon-coated wires and 53 ns for copper wires (not counting switch delays), or about 6 and 11 cycles for a 200 MHz machine. Thus, although all messages must incur this latency, it is not a very severe problem.

As for locality and bandwidth, if the average distance traveled in each physical dimension is one-half the length of the machine in that dimension, then the total distance traveled is about 3.6 meters instead of 8. This implies that only half of the wires are being used and that therefore we can expect about double the bandwidth. Higher locality implies further increases in bandwidth.

The bisection wire density is *not* the limiting factor for a 64K-node multibutterfly, as might be expected [Dal90a]. In the first stage, each splitter has $\frac{3}{4} \cdot 65536 \cdot 2 \cdot w$ wires crossing the first four-way split. For $w = 4$, this is 393,000 wires. There is an additional 393,000 wires for channels leaving processors in this volume. The cross-sectional area of these wires with $\rho = 1\,\mathrm{mm}^2$ is 0.79 square meters. The area of the two interior sides of the volume containing the splitter is $2 \cdot s \cdot \frac{s}{2} = s^2$, which for our final volume of 14 cubic meters is $(2.4)^2 = 5.8$ square meters. This is plenty of area for the crossing wires, even if we double their diameter or number. The board area for connections is also physically achievable: with 4 wires per channel, each radix-4 dilation-2 router connects to a total of about 64 network wires, while each processor connects to about 16.

Although wiring is not a physical limit, it remains the dominant cost of multiprocessors and the dominant power sink. Thus, topologies that lead to fewer and shorter wires and simpler wiring, such as a three-dimensional mesh, may be more cost effective. However, in general, topologies with plenty of long wires provide lower latency since messages can travel greater distances with fewer hops.

Longer wires generally increase the bisection bandwidth as well. For example, the bisection bandwidth of a three-dimension mesh is $\Theta(P^{\frac{2}{3}})$, while the bisection bandwidth

[14] The connections are not completely random—there are significant constraints on the wiring as a whole.

[15] The volume of a J-Machine node is about $64\,\mathrm{cm}^3$.

of a multibutterfly is $\Theta(P)$. Since machines must be built in three dimensions, the upper bound on bisection bandwidth is $\Omega(P^{\frac{2}{3}})$; however, we have already shown that the $\Theta(P)$ bisection bandwidth is physically achievable for a 64K-node multibutterfly. Thus, although a three-dimensional mesh may be more cost effective, higher performance can be achieved by using long wires. Express cubes and fat trees both exploit long wires for higher performance [Dal91, Lei85].

5.1 The Metabutterfly

Two severe problems with the multibutterfly are the complexity of wiring the early stages and the absence of locality. Keeping in the spirit of the multibutterfly, I propose the *metabutterfly* to address these issues.

The metabutterfly is a family of networks that consist of nested multibutterflies. The idea is best explained by example. A 64K-node network could be implemented as a 2-level 256-node metabutterfly. Each level is isomorphic to a 256-node multibutterfly and the innermost level is in fact a multibutterfly. In the outer level, each "node" consists of 256 processors, for the expected total of $256 \cdot 256 = 64$K processors. Each "channel" in the outer level consists of 256 channels grouped together, so that the bandwidth is matched to the size of the 256-processor node. Thus, each output from the top level is one "thick" channel consisting of 256 real channels.

These 256 real channels are fed into the next (and last) level; they are the 256 inputs to the 256-node multibutterfly that forms the second level. Note that a 2-level 256-node metabutterfly contains 256 256-node multibutterflies in the second level, one each output in the top level. The total number of wires and stages is the same as for a 64K-node multibutterfly; the only difference so far is the grouping of the channels.

This grouping greatly simplifies building the network, since there are fewer logically distinct endpoints. The channels can be routed together and the any of the 256 channels can go to any of the 256 inputs for the target node.

As for locality, suppose we take half of the inputs to a top-level node and short-circuit them directly to the second level that contains those 256 processors. The other half still enters the top level; this reduces the channels in the top level by a factor of two. Of the inputs to the second level, half come from the first level and half come from the processors in the local 256-processor cluster. This provides two-level locality: 1) messages to nodes in this cluster stay within the cluster, and 2) messages to nodes in other clusters take the full path through both levels. The channels need not be split in half, any split will work.

In general, p levels induce p locality classes. For example, a 3-level 64-node metabutterfly, which has $2^{3 \cdot 6} = 256$K nodes, might have half of the channels remain in the cluster, one quarter go up to the second level and on quarter go up to the top level.

Given that the metabutterfly simplifies wiring and provides locality, the next issue is its fault tolerance and congestion tolerance. First off, *the metabutterfly provides the same expansion as the multibutterfly on which it is based.* This is because each level is isomorphic to the base multibutterfly and each node is uniformly scaled up.

This is in fact significantly less expansion than a regular randomly-wired multibutterfly would have in the first stage. However, it is easy to show that the level of expansion in the metabutterfly is sufficient. Assuming each top-level node consists of 256 processors, partition each top-level node into 256 numbered classes. If we assign channels so that they always connect nodes in the same numbered class, we get 256 copies of a 256-node multibutterfly, one for each numbered class. Under this view, called the *planar view*, each plane has the exact same fault tolerance and performance properties of a 256-node multibutterfly, as well as the same number of components. From Section 4 we know that 256-node multibutterfly provides more than enough fault tolerance for the number of components it contains. (The components in other planes are irrelevant and thus need not be counted.) Thus, the fault tolerance provided by one plane is more than sufficient for the expected number of faults, and *the fault tolerance of the metabutterfly as a whole is thus also sufficient.*

The planar connection strategy is actually the worst case. Randomly routing between numbered classes provides better expansion; it is also far easier—any cable can go to any switch in the target node.

Finally, grouping the channels has two other benefits. First, to reduce wiring costs and component count, the channels in a group may multiplex a relatively small number of wires, perhaps one-fourth of that required for full wiring. Second, the group may share the high bandwidth provided by optical fiber. Neither of these options is available in regular multibutterflies, since the channels all have different logical endpoints.

Although the metabutterfly has several advantages over multibutterflies for large-scale machines, the metabutterfly may still not be the best network choice. Fat trees and express cubes are better asymptotically, provide locality, and can provably be built. The metabutterfly should be viewed simply as a proposal that addresses shortcomings in the multibutterfly while maintaining its essence.

6 Performance

There is plenty of evidence that adaptive routing improves the performance of networks under high congestion. However, there is no evidence that real systems will perform better using adaptive routing. There are two reasons that adaptive routing could do worse than simple, deterministic routing in practice.

6.1 Software Overhead

First, deterministic routing preserves message order, while adaptive routing does not. Thus, systems with adaptive routing must reorder messages in software for applications that require in-order delivery, such as pipelines, terminal I/O, and some languages [YSTH87, WBC+91].

The overhead for reordering depends upon the whether or not the network ensures delivery. Current multiprocessor networks, such as the CM-5 and the J-Machine, ensure delivery [ND92, L+92]. In this case, software reordering costs about 14 cycles, assuming that messages are *usually* delivered in order [BCH+92]. Although this seems insignificant, it is *more* than the total end-to-end software overhead for message passing on the J-Machine [NWD93]; other fine-grain machines such as *T should suffer similar penalties [NPA92].

Furthermore, the standard approach uses sequence numbers, which requires $\mathcal{O}(n)$ space per processor, or $\mathcal{O}(n^2)$ space total. The LSW protocol [LSW91] could be used on machines with synchronized clocks (or cycle counters [BKW93]), in which case the space is proportional to the number of actively communicating pairs. Another alternative is to always wait for an acknowledgment before sending a second message to the same processor, but this severely limits performance [CBH+92]. Both of these latter two approaches still take $\mathcal{O}(n^2)$ space in the worst case and will probably require at least as much overhead.

If the network may drop messages, which is common in distributed systems but very rare in multiprocessors, then the software must be prepared for retransmission. In this case, the cost of reordering is zero: it can be folded into the overhead for ensuring at-most-once delivery [CBH+92].

Thus, adaptive routing incurs additional software overhead compared with deterministic routing if and only if the network ensures delivery, which is the common case in multiprocessors today. Although this overhead is small, it is very significant on machines intended for fine-grain parallelism.

Software retransmission should become more common as machines get larger, so perhaps the extra overhead of adaptive routing due to message order will vanish. However, an interesting and important exception is the 16K-node CM-5, which guarantees delivery despite its size. It uses adaptive routing and thus does not ensure in-order delivery.

Finally, the cost of order preservation may not affect the primary applications of the machine, so this issue may not matter much in practice. Many applications do not depend on order, usually because they use higher level synchronization, such as barriers, to ensure correctness.[16] For those applications that do require order preservation, adaptive routing will increase the cost of message passing.

6.2 Cycle Time

The second advantage of deterministic routers is the speed achievable because of their simplicity: deterministic routers may be able to run at higher clock rates. In this case, adaptive routing performs worse under light load, but should still perform better under heavy congestion.

Although the network switches are simpler than the processor, they still affect the cycle time of the machine. In particular, it is common to run the network at multiple times the processor frequency. For example, the Alewife and the J-Machine both run the network at twice the speed of the processor [A+91, D+89]. The key advantage to running the network at higher speeds is the resulting reduction in wires and pins. For example, if the processor can feed 32 bits to the network per cycle, then running the network at four times the speed implies (roughly) one fourth the wires for the same bandwidth.[17]

The significant increase in complexity required for adaptive routing implies that the speed of deterministic routers might be much better. Either the cycle time might be lower or the network might run faster relative to the processor, perhaps three or four times faster compared to only a factor of two for adaptive routing. Alternatively, the adaptive router could have the same cycle time, but require more cycles to make the decision; one extra cycle should be sufficient and two seem certain to be enough.

The extra time required by adaptive routing is largely ignored in the adaptive routing literature; it is conspicuously absent from the analysis of the Chaos router [KS90, KS91]. Invariably, adaptive routing simulations assume that the cycle time and the decision time in cycles is the same for the adaptive router as it is for the deterministic router.

Since keeping the cycle time the same maintains throughput, I will assume that adaptive routing requires an extra cycle per hop. Thus, under high load, adaptive routing should still far outperform deterministic routing. However, under light load messages require an extra p cycles for p hops, which typically represents an increase of 25 to 50 percent. Agarwal shows

[16] The "pipe" mechanism in the Prelude language [WBC+91] addresses this issue by allowing programmers to incur the cost of order preservation only where their application requires it [CBH+92].

[17] In fact, the router is on the critical path for the J-Machine's MDP, although this is due largely to the robust I/O pads used for network connections. Point-to-point matched pads would eliminate more than 10 ns from this path [Nut].

that the expected number of hops for a toroidal bidirectional k-ary n-cube is $\frac{kn}{4}$ [Aga91]. Thus, with random traffic, the latency for a lightly loaded 16x16 torus would increase by 8 cycles, which would be significant in all of the adaptive routing simulations in the literature [DA92, CK92a, GN92, BGP92, KS90, KS91].

Planar adaptive routing [CK92a] addresses this issue to some extent by arguing for a simpler adaptive routing algorithm, although the algorithm presented is still significantly more complex than deterministic routing.

The fundamental relative complexity of adaptive routing leads to the following design principle:

Design Principle 4: Adaptive routers should be bimodal.

The key idea is that the router be designed from the beginning as consisting of two parts, a deterministic part and an adaptive part. The deterministic part is essentially identical to current deterministic routers; the adaptive part is much more complex. The two parts work in parallel: if the output chosen by the deterministic part is available, then the message is immediately sent along that path and the adaptive part gives up. If the chosen output is blocked, the deterministic part gives up and the adaptive part continues routing and completes in one or two cycles.

This principle leads to several nice properties:

1. The performance of the router under light load is nearly identical to that of a deterministic router, regardless of the complexity of the adaptive routing algorithm.

2. The performance of the router under heavy load achieves the full benefit of adaptive routing. In particular, note that the extra cycles required by the adaptive router have little impact on performance since they only occur during local congestion. In fact, if the message ends up blocking, the extra overhead is zero since the message would have spent those cycles blocked.

3. The fault tolerance of the bimodal router is exactly the same as that of its adaptive half, as is its freedom from deadlock. This is because the adaptive part always makes the decision when a message blocks.

4. The explicit bimodal nature may actually simplify the router since it removes pressure to speed up the control logic for adaptive part, although the area requirements are probably still higher than a router that performs pure adaptive routing.

5. One of the problems with fully adaptive routers is that the full destination must be known in order to make the decision, which presents a problem if the information does not fit in the first cycle's worth of bits.[18] By allowing the adaptive decision to take multiple cycles, we can ensure that all of the needed information is available.

6. In some simulations [GN92], deterministic routing outperformed adaptive routing under light load even assuming equally fast decision making. This is because the deterministic algorithms implicitly promote global load balancing—channels are equally loaded under random traffic. Adaptive routing is based on local decisions that may prove detrimental from a global viewpoint. Bimodal routers preserve this global balance under light load, which is the only time it is applicable.

[18] This is one of the motivations for the planar adaptive router algorithm [CK92a].

Given these properties, an adaptive router should be 1) nonminimal and fully adaptive, to maximize fault tolerance and high-load performance, and 2) bimodal, to maximize light-load performance and to separate the complexity of the adaptive routing decision from the critical path of the router.

7 Summary of Criticisms

7.1 Duato's Methodology

I provide little criticism of Duato's methodology; it is correct and fairly general. The only criticism is that (as used) it produces routers that revert to deterministic routing to avoid deadlock. This prevents these routers from ensuring delivery, even if there is a path from sender to target. Randomized misrouting does not have this property.[19] A proposed solution that fits within Duato's methodology is to revert to a partially adaptive algorithm based on the turn model [GN92]; this would greatly increase the likelihood of delivery. In practice, the reversion to deterministic routing can be made relatively unlikely, so this problem may have little impact on real systems.

7.2 The Chaos Router

There are two serious problems with the Chaos router (and many very nice things). First, it uses virtual cut-through routing rather than the more efficient and higher performance wormhole routing. Switching to wormhole routing only requires a new proof of freedom from deadlock; the probabilistic proof of freedom from livelock and the use of randomization would remain the same.

The second serious problem is the failure to deal with the slower routing decision implied by the complexity of adaptive routing compared with deterministic routing. All of her simulations assume that the decisions are equally fast. In a separate document [KS91], she hints at using a fast path to speed up the router, but the fast path is still more complicated than a deterministic router. A bimodal design should allow the Chaos router to achieve the impressive results predicted by simulations.

7.3 Multibutterflies

Despite optimal asymptotic fault tolerance, for medium-scale machines multibutterflies are not as tolerant of faults or congestion as k-ary n-cubes that use nonminimal routing. This is a novel result.

For large-scale machines, where the improved fault tolerance should appear, the multibutterfly becomes unrealistic to build, primarily because of the cost and complexity of the wiring. Thus, there is no domain in which the multibutterfly is the best choice for a network.

Another serious criticism of multibutterflies is their inability to exploit locality. Forcing all traffic to travel the maximum distance increases latency and reduces the achievable bandwidth for a given amount of wire.

The proposed metabutterfly simplifies wiring, exploits locality, and still provides expansion and sufficient fault and congestion tolerance.

[19] If one-way edge faults are allowed, even randomized misrouting cannot ensure delivery, since a message could enter a partition that has no return path.

8 Conclusions

This paper examined adaptive routing in multiprocessor networks, particularly k-ary n-cubes and multibutterflies. The first set of results is the analysis and simulation of multibutterflies that shows that multibutterflies are inferior to k-ary n-cubes for medium-scale machines, assuming that the k-ary n-cubes use nonminimal adaptive routing. Furthermore, the large-scale machines that might make multibutterflies superior are not realistically buildable. I also present the metabutterfly, which addresses two shortcomings of the multibutterfly: the wiring complexity and the inability to exploit locality.

The second set of results guides the design of adaptive routers. In particular, I propose four design principles:

1) Endpoints are the key to fault tolerance.
When networks become disconnected due to faults, it is usually because one node is partitioned off. Doubling the number of input and output channels in a 64-node multibutterfly tripled the percentage of faults that could be tolerated before the network became disconnected. The more paths in and out of a processors the more fault tolerant the network becomes. (This principle is hinted at in the literature.)

2) Edges should be bidirectional.
Using bidirectional edges doubles the number of paths in the network and thus greatly increases fault and congestion tolerance. Bidirectional k-ary n-cubes outperform multibutterflies under heavy congestion, while unidirectional k-ary n-cubes perform worse than multibutterflies when the number of channels exceeds about 45%. (This principle is fairly obvious but does not really appear in the literature.)

3) Routers should be nonminimal under duress.
Nonminimal adaptive routing far outperforms minimal adaptive routing, both in fault and congestion tolerance. However, since misrouting increases the distance a message must travel, it should only be used when all minimal options are ruled out. (This principle is not original.)

4) Adaptive routers should be bimodal.
Adaptive routing is fundamentally more complex than deterministic routing and thus requires more time for decisions. A bimodal router achieves the fast decision time of deterministic routers for light load, and achieves the dramatic improvement of adaptive routing under heavy load. Bimodal routers have the same fault tolerance and freedom from deadlock as their adaptive component. Furthermore, bimodal routers separate the complexity of adaptive routing from the critical path of the router, which promotes more complex, more effective adaptive routing. (This principle is original.)

Given these principles, it follows that adaptive routing should be used in multiprocessors and that the network should maximize the number of processor connections. The router should be bimodal, fully adaptive, and nonminimal under duress due to congestion or faults. These principles should allow adaptive routing to achieve excellent fault and congestion tolerance, while remaining competitive with deterministic routers in terms of light-load network performance.

References

[A+91] A. Agarwal et al. The MIT Alewife machine: A large-scale distributed-memory multiprocessor. In *Scalable Shared-Memory Multiprocessors*. Kluwer Academic Publishers, 1991.

[Aga91] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4), October 1991.

[BCH+92] E. A. Brewer, A. Colbrook, W. Hsieh, P. Wang, and W. E. Weihl. Pipes: Linguistic support for ordered asynchronous invocations. In *Proceedings of the 2^{nd} Workshop on Languages, Compilers, and Run-Time Environments for Distributed Memory Multiprocessors*, October 1992.

[BGP92] P. E. Berman, L. Gravano, and G. D. Pifarré. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. In *Proceedings of the 1992 Symposium on Parallel Architectures and Algorithms (SPAA '92)*, pages 3–12, June 1992.

[BKW93] E. A. Brewer, J. Kubiatowicz, and C. A. Waldspurger. Cycle counters: The time has come. Technical Report (to appear), Massachusetts Institute of Technology, 1993.

[CBH+92] A. Colbrook, E. A. Brewer, W. C. Hsieh, P. Wang, and W. E. Weihl. Pipes: Linguistic support for ordered asynchronous invocations. Technical Report MIT/LCS/TR-539, Massachusetts Institute of Technology, April 1992.

[CED92] F. Chong, E. Egozy, and A. DeHon. Fault tolerance and performance of multipath multistage interconnection networks. In T. F. Knight, Jr. and J. Savage, editors, *Advanced Research in VLSI and Parallel Systems 1992*. MIT Press, 1992.

[CK92a] A. A. Chien and J. H. Kim. Planar adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proceedings of the 19^{th} International Symposium on Computer Architecture (ISCA '92)*, pages 268–277, May 1992.

[CK92b] F. T. Chong and T. F Knight, Jr. Randomized multipath min architectures. In *Proceedings of the 1992 Symposium on Parallel Architectures and Algorithms (SPAA '92)*, June 1992.

[D+89] W. J. Dally et al. The J-Machine: A fine-grain concurrent computer. In G.X. Ritter, editor, *Proceedings of the IFIP Congress*, pages 1147–1153. North-Holland, August 1989.

[DA92] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 3(?), ?? 1992.

[Dal90a] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[Dal90b] W. J. Dally. Virtual-channel flow control. In *Proceedings of the* 17^{th} *International Symposium on Computer Architecture (ISCA '90)*, pages 60–68, May 1990.

[Dal91] W. J. Dally. Express cubes: Improving the performance of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, September 1991.

[DeH93] A. DeHon. Robust, high-speed network design for large-scale multiprocessing. Master's thesis, Massachusetts Institute of Technology, January 1993.

[DLD93] L. R. Dennison, W. S. Lee, and W. J. Dally. High-performance bidirectional signalling in VLSI systems. In *Proceedings of the 1993 Symposium on Integrated Systems (SIS '93)*, March 1993.

[DS87] W. J. Dally and C. L. Seitz. Deadlock free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[Dua91a] J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: Design methodologies. In *Proceedings of PARLE '91*, pages 390–405. Springer-Verlag, June 1991.

[Dua91b] J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: Theoretical aspects. In *Proceedings of the* 2^{nd} *European Distributed-Memory Computing Conference*, April 1991.

[Gel81] D. Gelernter. A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks. *IEEE Transactions on Computers*, C-30(10):709–715, October 1981.

[GLKA89] J. W. Goodman, F. J. Leonberger, S.-Y. Kung, and R. A. Athale. Optical interconnections for VLSI systems. In H. J. Caufield and G. Gheen, editors, *Selected Papers on Optical Computing*, pages 349–365. Springer-Verlag, 1989.

[GN92] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proceedings of the* 19^{th} *International Symposium on Computer Architecture (ISCA '92)*, pages 278–287, May 1992.

[Gra90] J. Gray. A census of tandem system availability between 1985 and 1990. *IEEE Transactions on Reliability*, 39(4):409–418, October 1990.

[Gru89] D. C. Grunwald. *Circuit Switched Multicomputers and Heuristic Load Placement*. PhD thesis, University of Illinois at Urbana-Champaign, September 1989.

[Gun81] K. D. Gunther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Transactions on Communication*, COM-29(4):512–524, April 1981.

[HLN89] J. Hastad, T. Leighton, and M. Newman. Fast computation using faulty hypercubes. In *Proceedings of the* 21^{st} *ACM Symposium on the Theory of Computing*, pages 251–263, May 1989.

[Hsi90] W. Hsieh. *The Optical Alternative for Digital System Interconnect*, 1990. Area Exam Paper.

[KK79] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, October 1979.

[Kni89] T. F. Knight, Jr. Technologies for low latency interconnection switches. In *Proceedings of the 1989 Symposium on Parallel Architectures and Algorithms (SPAA '89)*, pages 351–358, June 1989.

[Kon91] S. Konstantinidou. *Deterministic and Chaotic Adaptive Routing in Multicomputers*. PhD thesis, University of Washington, May 1991.

[Kon92] S. Konstantinidou. Priorities in nonminimal, adaptive routing. Technical Report RJ 8592, IBM Almaden Research Center, January 1992.

[KS90] S. Konstantinidou and L. Snyder. The chaos router: A practical application of randomization in network routing. In *Proceedings of the 1990 Symposium on Parallel Architectures and Algorithms (SPAA '90)*, pages 21–30, July 1990.

[KS91] S. Konstantinidou and L. Snyder. Chaos router: Architecture and performance. Technical Report RJ 8447, IBM Almaden Research Center, November 1991.

[KU91] S. Konstantinidou and E. Upfal. Experimental comparison of multistage interconnection networks. Technical Report RJ 8451, IBM Almaden Research Center, November 1991.

[L+92] C. E. Leiserson et al. The network architecture of the Connection Machine CM-5. In *Proceedings of the 1992 Symposium on Parallel Architectures and Algorithms (SPAA '92)*, pages 272–285, June 1992.

[Lei85] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[LH91] D. H. Linder and J. C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1):2–12, January 1991.

[LHM+89] A. L. Lentine, H. S. Hinton, D. A. B. Miller, J. E. Henry, J. E. Cunningham, and L. M. F. Chirovsky. Symmetric self-electrooptic effect device: Optical set-reset latch, differential logic gate, and differential modulator/demodulator. *IEEE Journal of Quantum Electronics*, 25(8):1928–1936, August 1989.

[LLM90] T. Leighton, D. Lisinski, and B. Maggs. Empirical evaluation of randomly-wired multistage networks. In *Proceedings of the 1990 International Conference on Computer Design: VLSI in Computers & Processors (ICCD '90)*, October 1990.

[LM92] F. T. Leighton and B. M. Maggs. Fast algorithms for routing around faults in multibutterflies randomly-wired splitter networks. *IEEE Transactions on Computers*, 41(5):578–587, May 1992.

[LSW91] B. Liskov, L. Shrira, and J. Wroclawski. Efficient at-most-once messages based on synchronized clocks. *ACM Transactions on Computer Systems*, 9(2), May 1991.

[Mai88] J. N. Mailhot. *A Comparative Study of Routing and Flow-Control Strategies in k-ary n-cube Networks*. Massachusetts Institute of Technology, May 1988.

[Mes90] D. G. Messerschmidt. Synchronization in digital system design. *IEEE Journal on Selected Areas in Communications*, 8(8):1404–1419, October 1990.

[MS80] P. M. Merlin and P. J. Schweitzer. Deadlock avoidance in store-and-forward networks—i: Store-and-forward deadlock. *IEEE Transactions on Communication*, COM-28(3):345–354, March 1980.

[ND92] P. R. Nuth and W. J. Dally. The J-Machine network. In *Proceedings of the 1992 International Conference on Computer Design: VLSI in Computers & Processors (ICCD '92)*, October 1992.

[Noa] M. Noakes. Personal Communication.

[Noa92] M. Noakes. Radix sort on the J-Machine. Technical Report MIT/AI/CVA Memo 47, MIT Concurrent VLSI Architecture Group, December 1992.

[NPA92] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *t: A multithreaded massively parallel architecture. In *Proceedings of the* 19^{th} *International Symposium on Computer Architecture (ISCA '92)*, May 1992.

[Nut] P. Nuth. Personal Communication.

[NWD93] M. D. Noakes, D. A. Wallach, and W. J. Dally. The J-Machine multicomputer: An architectural evaluation. In *Proceedings of the* 20^{th} *International Symposium on Computer Architecture (ISCA '93)*, May 1993.

[PLF+91] R. Paturi, D.-T. Lu, J. E. Ford, S. C. Esener, and S. H. Lee. Parallel algorithms based on expander graphs for optical computing. *Applied Optics*, 30(8):917–927, March 1991.

[Rag89] P. Raghavan. Robust algorithms for packet routing in a mesh. In *Proceedings of the 1989 Symposium on Parallel Architectures and Algorithms (SPAA '89)*, pages 344–350, June 1989.

[TMC92] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary*, January 1992.

[Tou80] S. Toueg. Deadlock- and livelock-free packet-switching networks. In *Proceedings of the* 12^{th} *ACM Symposium on the Theory of Computing*, pages 94–99, 1980.

[TU79] S. Toueg and J. D. Ullman. Deadlock-free packet-switching networks. In *Proceedings of the* 11^{th} *ACM Symposium on the Theory of Computing*, pages 89–98, 1979.

[Upf89] E. Upfal. An O(log n) deterministic packet routing scheme. In *Proceedings of the* 21^{st} *ACM Symposium on the Theory of Computing*, pages 241–250, May 1989.

[WBC+91] W. E. Weihl, Eric Brewer, Adrian Colbrook, Chrysanthos Dellarocas, Wilson Hsieh, Anthony Joseph, Carl Waldspurger, and Paul Wang. PRELUDE: A system for portable parallel software. Technical Report MIT/LCS/TR-519, Massachusetts Institute of Technology, October 1991.

[YJ89] J. Yantchev and C. R. Jesshope. Adaptive, low-latency, deadlock-free packet routing for networks of processors. *IEE Proceedings*, 136 (Part E)(3), May 1989.

[YSTH87] A. Yonezawa, E. Shibayama, T. Tokada, and Y. Honda. Modelling and programming in an object-oriented concurrent language abcl/1. In A. Yonezawa and M. Tokoro, editors, *Object-Oriented Concurrent Programming*, pages 55–90. MIT Press, 1987.