

Design and Performance of Multipath MIN Architectures

Frederic T. Chong and Thomas F. Knight, Jr.
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

In this paper, we discuss the use of *multipath* multistage interconnection networks (MINs) in the design of a fault-tolerant parallel computer. Multipath networks have multiple paths between any input and any output. In particular, we examine networks with either the property of *expansion* or *maximal-fanout*. We present an $\Omega(n \log n)$ lower time bound for a worst-case permutation on deterministic maximal-fanout networks. We further show how a randomized approach to maximal-fanout avoids the regularity from which this worst case arises.

Unlike most previous work, we examine systems which can tolerate node failure and isolation. We describe mechanisms for fault identification and system reconfiguration. In reconfiguring a faulty system, a naive approach is to preserve processing power by maximizing the number of processing nodes left in operation. However, our results show that the synchronization requirements of applications make it critical to eliminate nodes with poor network connections. We find that a conservative *fault-propagation* algorithm for reconfiguration, adapted from work by Leighton and Maggs [LM92], performs well for all of our multipath networks. We also address some practical issues of network construction and present performance simulations based upon the MIT Transit architecture [DeH90]. Simulation results for 1024 node systems demonstrate that multipath networks, reconfigured with our fault-propagation algorithm, perform well not only in theory, but also in practice. In fact, our systems suffer only a small decrease in performance from network faults; the degradation is linear in the percentage of network failure.

Acknowledgments: This research is supported by an Office of Naval Research Graduate Fellowship and the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

To appear in *Symposium on Parallel Algorithms and Architectures*, San Diego, California, June 1992.
ACM.

Large-scale parallel multiprocessing appears to be the only technique for constructing teraflop to petaflop performance machines. These machines will require to high-performance processors to solve difficult computational problems such as tertiary protein conformation, weather prediction, and common-sense understanding. Two critical issues in the design of such machines, *interconnection topology* and *reliability*, are addressed in this paper. In thousand- to million-processor architectures, we must assume that at any given time some significant fraction of the processors, interconnection components, wiring, and power will be faulty. The achievable performance of such systems is limited by the latency of the communication, the process synchronization requirements, and the performance of the processor array and interconnect under faulty conditions.

In particular, we describe the simulated performance and design principles for the wiring of low-latency fault-tolerant multistage interconnection networks (MINs). Such multistage networks can provide nearly non-blocking interconnect with latency limited primarily by wire propagation delay. Very large networks, constructed in the fat-tree topology [Lei85] [DeH91a], use these MINs as basic building blocks, achieving high interconnection bandwidth. Such networks form the wiring topology of the Thinking Machines CM5 processor array [Th91].

The key factor in providing fault tolerance in MINs is the addition of multiple independent routing paths within the network. With independent routes, failures of components, wiring, or power in one path may leave a second path undamaged. There are many choices in the construction of such multipath networks, differing in detailed wiring topology and in the strategies used in recovery from router or wiring failures. The comparative evaluation of these wiring techniques and recovery strategies is the subject of this paper.

In this paper, we use the term *node* to denote the collection of processor and memory hardware connected to network endpoints. The term *routers* refers to the switches which comprise the networks. In Section 2, we describe three multipath networks and perform some worst-case analysis on one of them. We describe the architecture on which our simulations are based in Section 3. In Section 4, we discuss practical issues of network construction relating to network input and output. We continue by developing our model of network loading in Section 5. We present our simu-

lation results in Section 6. Finally, we conclude with a discussion of processor node fault tolerance in Section 7, and of dynamic system partitioning in Section 8.

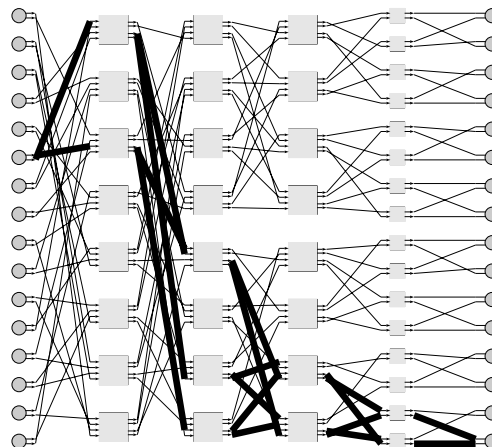
In this section, we describe three networks: the randomly-wired multibutterfly, the deterministic maximal-fanout network, and the randomized maximal-fanout network. We present an $\Theta(n \log n)$ lower time bound for a worst-case permutation on deterministic networks and show how a randomized approach to maximal-fanout avoids this worst case.

To achieve greater tolerance to component failures, we *interwire* our networks to provide multiple paths between each pair of endpoints. To understand the concept of interwiring, it is helpful to view routing through a MIN as a sorting function through equivalence classes of fewer and fewer routers. Consider a radix-2 network, where *radix* refers to the number of logical directions switched by each router. The canonical butterfly network is a well-known example of such a network. The routers in the first stage of the network belong to a single equivalence class. A message starts in this class and is routed to one of two classes in the second stage. In each subsequent stage, each message is routed to one of two classes, each containing half the number of routers as the classes in the current stage. Consequently, a message proceeds through the network, halving the number of possible destinations with each stage, until it has reached a class of size one at the end of the network. This last class is the message's destination.

In general, a message is *self-routed* by using a message header, $h = (h_1, h_2, \dots, h_n)$, where the h_i 's represent individual bits in the binary representation of i . There are $\log_2 n$ bits in h , where n is the number of stages in the network, and $\log_2 n$ equals the radix of the routers. At stage i , for $0 \leq i < n$, bits h_{i+1} through h_n are used to select between equivalence classes in the next stage. The header, h , is the same from every source to any particular destination, and is generally also used to name, or *address*, that destination.

The wiring scheme of a network must provide each router in an equivalence class with connections to the appropriate equivalence classes in every logical direction. However, our multipath networks are constructed from *dilated* routers. A dilated router has more than one output in each logical direction. The examples given in this section use a dilation-2 router, one which has two outputs in each logical direction. Interwiring refers to the practice of connecting these redundant outputs to different routers within the appropriate class. Bassalygo and Pinsky [BP74] used such interwiring to construct the first non-blocking networks of size $n \times n$ and depth $\log_2 n$. On-line algorithms for non-blocking routing are described in [ALM90]. Other work on interwiring multipath networks can be found in [CS82] [RK84].

The first network we examine is the randomly-wired multibutterfly, which randomly interwires within equivalence classes. Upfal first introduced the term *multibutterfly* in [Upf89]. With high

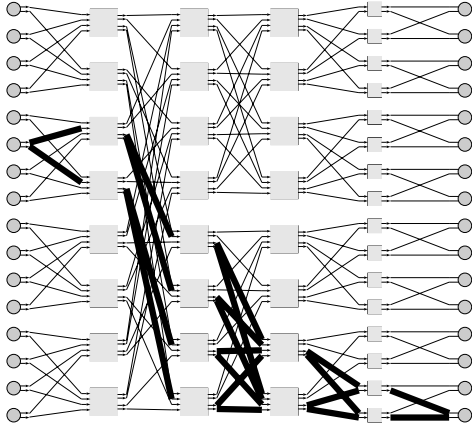


A randomly-wired four-stage multibutterfly connecting 16 endpoints. Each component in the first three stages is a radix-2, dilation-2 router. To prevent any unique critical paths between endpoints, the last stage is composed of radix-2, dilation-1 routers (see Section 4.2). The multiple paths between a selected pair of endpoints are shown in bold.

Figure 1: Randomly-Wired Multibutterfly

probability, random interwiring gives networks the property of *expansion*. Formally, an expansion property guarantees that any subset of k components from one M -input stage must connect to at least k components in the next stage, where $k \leq M$ and $M \geq 1$. Multistage networks with the property of expansion have been shown, in theory, to possess substantial fault tolerance and performance [Upf89] [LM92]. Our random interwirings, shown in Figure 1, are based upon the wiring presented by Leighton and Maggs [LM89].

The other two networks we shall examine have the property of *maximal-fanout* rather than that of expansion. Maximal-fanout guarantees that the paths between any two endpoints will use as many distinct routers as possible. In particular, a network is said to have maximal-fanout up to stage s , if, for every set, S , of paths between any particular pair of endpoints, and s , includes distinct routers at stage s . d is network dilation, r is network radix, n is the number of network stages, and f_s denotes, at stage s , the size of the tree of alternate paths, the *fanout-tree*, extending from s to n . f_s represents the size of the routing equivalence classes at stage s . In other words, the maximum number of distinct routers in S at stage s is limited by the sizes of both the fanout tree and the routing equivalence classes. Given these constraints, maximal-fanout can only extend to the middle of any network where radix equals dilation. However, our simulated architecture, to be discussed in Section 3, uses radix-4, dilation-2 networks in which fanout extends approximately two-thirds into the network. After maximal-fanout ends, the wiring of



A deterministic, maximal-fanout, four-stage network connecting 16 endpoints. Each component in the first three stages is a radix-2, dilation-2 router. To prevent any unique critical paths between endpoints, the last stage is composed of radix-2, dilation-1 routers (see Section 4.2). The multiple paths of the fanout-tree between a selected pair of endpoints are shown in bold.

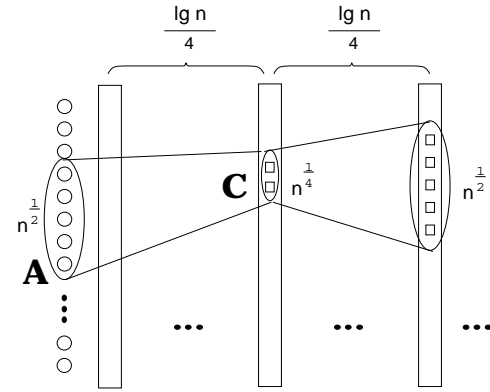
Figure 2: Deterministic, Maximal-Fanout Network

the remaining stages of the network is largely constrained by the sorting function of the network. We use isomorphic wirings for these remaining stages of all our multipath networks.

The deterministic network, presented earlier in [CED92] and shown in Figure 2, uses a regular butterfly topology to achieve maximal-fanout. Instead of using every output as a separate logical direction, the dilation of 2 in our network allows us to use pairs of outputs for each logical direction. We use these output pairs to create the exponential fanout-tree of paths between each pair of endpoints. Note that the multiple paths, shown in the maximal-fanout network of Figure 2, reach 4 routers in the second stage as compared to the 3 routers reached by paths in the multibutterfly of Figure 1.

Maximal-fanout increases fault tolerance and performance between any pair of *single* endpoints. However, it sacrifices expansion, which has been proven to have good fault tolerance and performance between *groups* of endpoints. We will compare these properties empirically in Section 6.

Unfortunately, the regularity of the deterministic network is vulnerable to worst-case permutations. For simplicity, we will first examine a radix-2, dilation-2 network. This network uses a regular wiring which is topologically equivalent to a 4-ary butterfly. The difference is that the 4 physical outputs of each dilated router are grouped into 2 pairs of logical outputs. The multiple paths resulting from this dilation avoid the single points of congestion a regular butterfly suffers on permutations such as bit-reversal. However, we can still construct a worst-case permutation for this multipath network which results in congestion in groups of routers.



The above figure illustrates the construction in the proof of Theorem 1. $n^{1/2}$ processor nodes are routing a worst-case permutation to $n^{1/2}$ destinations. While $n^{1/2}$ routers are available in stage i , there are only $n^{1/4}$ routers in stage $i+1$ through which the $n^{1/2}$ messages may pass. Therefore, the permutation takes at least $n^{1/4}$ time to route.

Figure 3: Worst-Case Congestion in the Deterministic, Maximal-Fanout Network

There exists a permutation which takes $\Omega(n^{1/4})$ time, in router cycles, to route on a radix-2, dilation-2 deterministic maximal fanout network.

We want to construct a permutation in which nodes in a set A , are each sending a message to a unique member of a set of nodes, B . Further, we want a permutation which forces all the messages to go through some small set of switches, C , at some point in the network.

Recall that, when radix equals dilation, maximal-fanout extends to the middle network. Let us examine the paths from a particular node a to a set of output nodes, B . Let A include all nodes with addresses with the first half of their bits equal to some constant, a . There are $n^{1/2}$ nodes in A and $n^{1/2}$ routers in the middle stage which can reach nodes in B .

Since the network has maximal-fanout, we know that node a reaches all of these $n^{1/2}$ routers in the middle stage through a binary tree. Let us look at the $(\lg n)/4$ -th stage of the network. There are $n^{1/4}$ routers, C , of the fanout-tree from a to B in this stage. Each one of these routers in C reaches $n^{1/4}$ nodes on the input side. However, the topology of the 4-ary butterfly is such that this set of the $n^{1/4}$ nodes, A , is the same for each of the routers in C . This is the key to the proof.

Consequently, if each of the $n^{1/2}$ nodes in A is trying to reach a unique member of the $n^{1/2}$ nodes in B , all $n^{1/2}$ messages must go through the $n^{1/4}$ routers in C . Therefore, such a permutation would take at least $\Omega(n^{1/4})$ cycles to route. Figure 3 illustrates the paths involved in such a permutation.

In fact, the further maximal-fanout extends into the network, the worse our bound. Recall that our simulations use radix-4, dilation-2 networks in which maximal-fanout extends roughly two-thirds into the network. This results in a lower bound closer to $\frac{2}{3}$. However, all of the preceding asymptotic analysis, while interesting for future scalability, assumes networks larger than those we shall be examining in detail. In fact, simulations of 1024-node, deterministic networks do not show significant worst-case behavior.

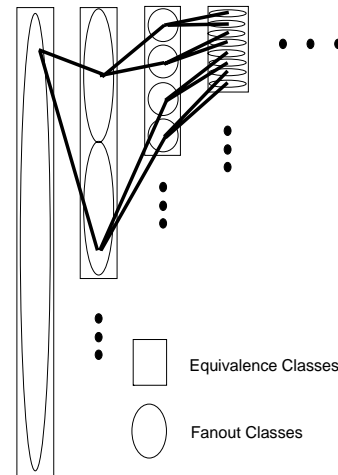
Fortunately, the worst-case permutation in the previous analysis can be avoided by a randomized fanout scheme. Figure 4 illustrates how randomly chosen connections between the appropriate *fanout classes* produce maximal-fanout for each pair of nodes. Each stage of the network is divided into routing equivalence classes. In turn, each routing equivalence class in stage i is divided into 2^i fanout classes, where i is the number of stages in the network. To wire a pair of dilated outputs of a router, (i, j) , select the appropriate routing equivalence class for the pair's logical direction. Then select fanout class numbers k and l within that routing class, where k is the fanout class containing the output pair of (i, j) . Randomly select a free input port in each of the two fanout classes and wire one port of the output pair to each. Since the children of each port in each fanout-tree are randomly selected from disjoint fanout classes, this process ensures that fanout-trees will have physically distinct components. This process continues only until the last stage of maximal-fanout — where fanout classes are of size 1. Note that the randomization in this network avoids regularity, but does not produce expansion.

We now describe the system architecture modeled in the simulations presented in this paper. The system is based upon an architecture under construction by the MIT Transit project.

The networks simulated use a circuit-switched routing component based upon the RN1 [MDK91], a custom VLSI chip, and the RN2 [DeH91b], a chip under design. Each chip can act either as a single 8-input, radix-4, dilation-2 router, or as two independent 4-input, radix-4, dilation-1 routers.

To aid the routing of messages, each component will have a pin dedicated to calculating flow control information according to the following blocking criterion taken from Leighton and Maggs in [LM92]. A router is *blocked* if it does not have at least one unused, operational output port in each logical direction which leads to a router which is not blocked. To route a message, a router attempts to choose a single output port by first looking at unused ports, and second eliminating any ports which are blocked. If no unique choice arises — all ports unused, but all unblocked or all blocked — then the router randomly decides between ports.

Each chip also incorporates a serial *Test Access Port* (TAP) which accesses *IEEE boundary scan* facilities [Com90]. These



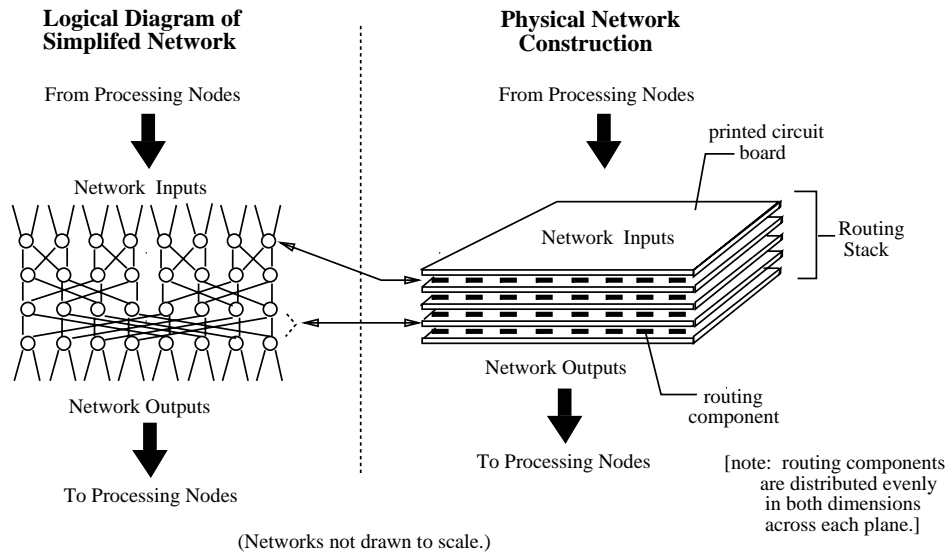
This figure shows how to achieve maximal-fanout while avoiding regularity. The routers shown are radix-2 and dilation-2. For each stage, we divide each routing equivalence class into an exponentially larger number of fanout classes until fanout classes will no longer fit. Random wirings are chosen between appropriate fanout classes to form fanout-trees. The disjoint nature of fanout classes ensures that fanout-trees will have physically distinct components.

Figure 4: Randomized Maximal-Fanout

ports, in turn, are connected together in a diagnostic network controlled by a *Boundary Scan Controller* (BSC) which can provide in-operation diagnostics and chip reconfiguration [DeH92].

Each attempt to send a message through the network automatically receives status information from each router involved. If routing errors occur, an accumulation of this status information can be used to identify suspect chips. To test a chip for faults, the BSC first isolates the chip from the network by disabling all ports on other chips which lead to the suspect chip. The BSC can then run a test pattern through the chip and identify any failures. It then disables any of the ports which can no longer function. Finally, it re-enables any ports from other chips which lead to still-functioning ports on the suspect chip. Similarly, interchip wiring may be tested for continuity, isolation, and electrical performance.

In order to decrease wire-lengths and machine size, the Transit architecture [DeH90] uses the three-dimensional stack packaging scheme shown in Figure 5. Vertical connections are made via button board connectors. Routing components are arranged in columns directly above one another. This suggests an economical method of wiring the diagnostic network. Columns of chips can have their boundary scan pins vertically connected. These columns can then be connected to some number of Boundary Scan Controllers.



The Transit packaging consists of a three-dimensional stack of alternating router and interconnect boards. The boards are connected with button board connectors. Router boards contain RN1 routing components and interconnect boards provide the appropriate wiring between these components.

Figure 5: Transit Packaging (Diagram courtesy of André DeHon)

Having described our simulated architecture, we turn to some practical issues of network construction. Much of the fault tolerance and routing behavior of our networks is dominated by the first and last stages. Although multipath networks provide multiple paths between any two nodes, these paths can only use a large number of physically distinct routers towards the middle of the network. Near the nodes, these paths must concentrate towards their specific sources and destinations. This concentration is most severe in the first and last stages, where each node only has a small number of connections to the network. We now describe methods of coping with this limited network I/O. Many of these ideas were initially explored in [DKM91].

The wiring of the nodes to the inputs of the first-stage routers involves a tradeoff between smoothness of degradation and mean time between machine reconfigurations. Each node has 2 inputs into distinct routers of the first stage of our networks. This means that each first-stage router has 8 nodes connected to its inputs. For nodes with inputs randomly wired to distinct routers of the first stage, let t be the mean time between *isolation* of any node's inputs. A node is isolated if router failures prevent it from communicating with any other nodes.

Now consider a wiring in which sets of 8 nodes are wired to 2 routers, each node connected to both routers. The time between any node loss is decreased, but if we lose nodes, we lose

8 at a time. To a first approximation, the mean time to isolation of a single node is also t . Therefore, the mean time to loss of each cluster of 8 is approximately $t/8$. In clustering the nodes, we have also clustered the node failures in time. Instead of having one node isolation every t cycles, we have approximately 8 node isolations every t cycles. In general, we can cluster processors to r routers in a regular butterfly pattern such that we lose r processors at a time, for r radix, and r dilation. This results in a mean time to cluster isolation of approximately t/r .

The desirability of clustering depends on several factors. If the system can not tolerate any node isolation, maximal clustering will give the greatest expected time to system failure. If the system can reconfigure itself to accommodate node isolation, then the reconfiguration scheme becomes an issue. If reconfiguration is dynamic, then the smoother node loss may be easier to handle. If it is static, then it may be desirable to run for as long as possible, then stop the machine and reconfigure for a large number of isolated nodes.

The amount of clustering will also affect each cluster's bandwidth into the network. However, in simulations of both worst-case and uniform message traffic, the bandwidth through the rest of the network does not significantly exceed what the first stage routers provide to each cluster.

The size of each routing equivalence class in the last stage of our network is one chip. Unfortunately, if each chip

were assigned both network connections to each node, the network would have a single point of failure. To avoid this difficulty, we split each chip into two logical routers. Now each routing equivalence class has two logical routers, which we ensure are assigned to physically distinct routers. Consequently, it takes at least two router failures in the last stage to isolate any node. Figures 1 and 2 illustrate this last-stage wiring.

Before we present our simulation results, we need to establish our method of network loading. We construct a specific task, representative of shared-memory applications, by examining message length, message frequency, and time between processor synchronizations. A shared-memory model assumes a uniform address space for all memory in the system. We model a distributed-memory system which supports this abstraction with coherent caches. Each processor has a local memory and cache. Network traffic occurs for two reasons. First, when data reads or writes are neither cached nor in local memory a message must be sent to a processing node whose local memory contains the data. Second, coherency messages are necessary to keep the data in caches consistent.

Previous work [CED92] examined several shared-memory applications by looking at message traffic taken from [CFKA90]. It was found that traffic consisting uniformly of 24-byte messages was representative of effects observed from those shared-memory applications. Those applications used barrier-synchronization, manipulated 16-byte cache lines and maintained cache coherency. Barrier-synchronization refers to points in an application where all processors must arrive at known points of execution. The size of cache lines affects the size of messages for remote reads or writes.

To derive the frequency of message generation, several “reasonable” parameter values were chosen. We have found that our results are not overly sensitive to loading, so rough figures are adequate. The program code for each processor is assumed to be resident in local memory. Consequently, only data references will result in non-local memory references. We assumed a data cache miss rate of 15 percent. For each data read or write, we get 0.15 misses per processor cycle. We assumed that 50 percent of the references are to local memory, which gives us 0.075 references to non-local memory per processor cycle. With a 50 MHz processor and the RN1 part running at better than 100MHz, we have 2 router cycles per processor cycle. This gives us 0.0375 non-local memory references per router cycle. Adding an additional 10 percent to account for cache coherency messages, we end up with approximately 0.04 messages per router cycle.

We also examine time between synchronizations, or, equivalently, application grain size. A grain size representative of applications studied is 10,000 cycles, or messages. Altogether, our performance metric is the time to route the following task: all processors in the system must each send 400 24-byte messages at a rate of 0.08 messages per active processor cycle. We assume that each processor can have up to 4 threads, or tasks, each with an outstanding message, before stalling.

Note that our task explicitly models barrier synchronization. Other styles of synchronization may involve smaller processor groups, but may also tend to synchronize these groups more often. In any case, it is important to model the synchronization requirements of an application. For our purposes, barrier synchronization incorporates an appropriate component of these requirements into our performance metric. We shall see in Section 6 that synchronization plays a major role in performance degradation. This degradation occurs when network failure results in a small number of nodes with particularly poor communication bandwidth.

Let us take one more look at our numbers. Since messages are 24 bytes long, we are basically running our network at 1 byte per router cycle, or 100 percent. If the message rate were any higher, the processors would just be stalled more often, and the network loading would not really change. Note that our analysis assumes low latency message handling, a concept demonstrated in the J-Machine [D 92]. If, as with many commercial and research machines, there exists a high latency for message handling, the latency induces a feedback effect which prevents full utilization of the network [JA92]. Although cache miss and message locality numbers are open to debate, we observe that the technological trends of multiple-issue processors and wider cache lines will only increase demands on the network. However, performance results presented in this paper were also verified to be qualitatively unchanged under network loading half of that used here.

In this section, we describe two methods of system partitioning and compare their performance. Previous work [LLM90] [CED92] has concentrated on the fault performance of multipath networks in which every endpoint could communicate with every other endpoint. In a more realistic view, we now examine the performance of systems which can tolerate network failures which isolate endpoints. Once such failures occur, a system must have some partitioning algorithm for determining *live nodes* with which to continue operation.

As was discussed in Section 4, the dominant effect of network failures is the isolation either of node outputs into the first stage of the network, or of node inputs from the last stage. Any partitioning algorithm must recognize such *I/O-isolation* of nodes and remove them from the set of live nodes.

Before we begin, we present a few notes about the data in figures 6 and 8: fault tolerance and performance results were obtained through Monte Carlo simulation of uniformly distributed router failures. Performance results reflect the time to route a task, as specified in Section 5, which simulates a barrier-synchronized shared memory application. Note that we are interested in steady-state performance — the time to route the task after fault reconfiguration has already occurred. Lastly, the granularity of our task is assumed to allow it to be evenly redistributed to the live nodes.

Initially, we focused on retaining the largest number of processors possible by only excluding I/O-isolated nodes from the set of live

nodes. However, using only the I/O-isolation criterion, many live nodes may not be able to directly communicate with each other. To overcome this problem, we used a *multi-hop protocol*, which allows a message to visit intermediate processing nodes before reaching its final destination. Such a scheme allows any node to communicate with another as long as they are connected in the transitive closure of node-to-node connections. The multi-hop protocol is an end-to-end variation of a similar scheme which was proposed in [VR88].

Figure 6 summarizes system fault tolerance and node loss for 1024-node (5-stage) systems based on randomly-wired networks. Figure 6(A) shows the probability the system can tolerate various levels of faults in the network given no reconfiguration, I/O-isolation, and I/O-isolation combined with the multi-hop protocol. Recall that in order for the system to remain in operation, all live nodes must be able to communicate with each other. Although the multi-hop protocol substantially increases system fault tolerance, we can see that it only provides reliable operation when network failure is below 15 percent. For our fault tolerance study we have performed an connectivity and transitive closure test for each Monte Carlo trial, for an node system. In an actual system, such an expensive test may not be practical. Therefore, practical systems using the multi-hop protocol may be limited to the reliable region of fault tolerance, under 15 percent of network failure in the case of our 1024-node systems.

Figure 6(B) shows the average percentage of node loss, under the multi-hop protocol, for *only the fraction of networks which were usable in Figure 6(A)*. In other words, trials in which the system could not provide communication between any pair of live nodes are not counted. Results for maximum fanout networks are similar to those of the multibutterfly.

Unfortunately, a small number of “weak” nodes, those with very poor connections to some other nodes, are still live. As results will show, synchronization requirements cause these weak nodes to significantly degrade performance.

Since weak nodes degrade performance, a more conservative criterion for choosing live nodes should result in better performance. Although fewer nodes are available to perform the task, we will no longer be limited by the speed of some of the weakest excluded nodes.

Leighton and Maggs proved in [LM92] the following theorem:

No matter how an adversary chooses faults in an n -input, n -output network with expansion, there exist at least k inputs and k outputs between which any permutation can be routed in $O(n/k)$ router cycles.

It turns out that we can use the proof of this theorem to construct an algorithm to select live nodes. The proof uses a stricter definition of I/O-isolation which includes nodes with inputs or outputs isolated by routers from the entire network, not just the first or last stages. However, due to the dominant effects of the first and last stage in the networks of the size we examined, we

- 1 Begin with all nodes live.
- 2 Determine I/O-isolated nodes and remove them from the set of live nodes.
- 3 Each faulty router leading to at least one live node is declared to be blocked. Propagate blockages from the outputs to the inputs according to the definition of blocking given in Section 3.
- 4 If all of a node’s connections into the first stage of the network lead to blocked routers, remove the node from the set of live nodes.

The above algorithm combines the definition of blocked routers, given in Section 3, with the concept of I/O-isolation to make a conservative choice in system reconfiguration.

Figure 7: Fault-Propagation Algorithm for Reconfiguration

find it both sufficient and practical to only look at those stages for I/O-isolation. The resulting *fault-propagation* algorithm is shown in Figure 7.

Figure 8(A) compares, for a 1024-node randomly-wired multi-butterfly system, the average node loss of the fault-propagation algorithm to that of the multi-hop protocol. For the multi-hop protocol, the sharp increase in node loss above 15 percent network failure reflects the trials in which the multi-hop protocol can not provide complete connectivity of live nodes. In contrast to Figure 6(B), these trials have been averaged in as 100 percent node loss. Although it was designed for networks with expansion, fault-propagation appears to also work well for other multipath MINs such as the maximal-fanout networks. Average node loss for maximal-fanout networks is basically indistinguishable from that of the multibutterfly.

Although the difference between average node loss under the multi-hop protocol and the fault-propagation algorithm is imperceptible for network failures of less than 15 percent, the handful of weak nodes eliminated, coupled with the overhead of the multi-hop protocol, make the fault propagated systems perform substantially better than the multi-hop systems. In fact, Figure 8(B) shows that the performance curves of the fault-propagated systems are nearly linear. For such a small network size, this is a somewhat surprising validation of asymptotic predictions implied by Theorem 2. Also of interest in Figure 8(B), the randomized, maximal-fanout network performed just as well as the multibutterfly.

Although we are primarily concerned with the steady-state performance of the system after reconfiguration, it is important to discuss the feasibility of tolerating node failure or isolation. Several approaches are available: checkpointing, redundant processors, and evacuating state.

Checkpointing computation is a well-known method of tolerating processor loss. Secondary storage devices would be attached to some of the network’s endpoints. There are various types of checkpointing [SY85] [SW89] involving various levels of optimism. However, the time overhead of checkpointing in a parallel

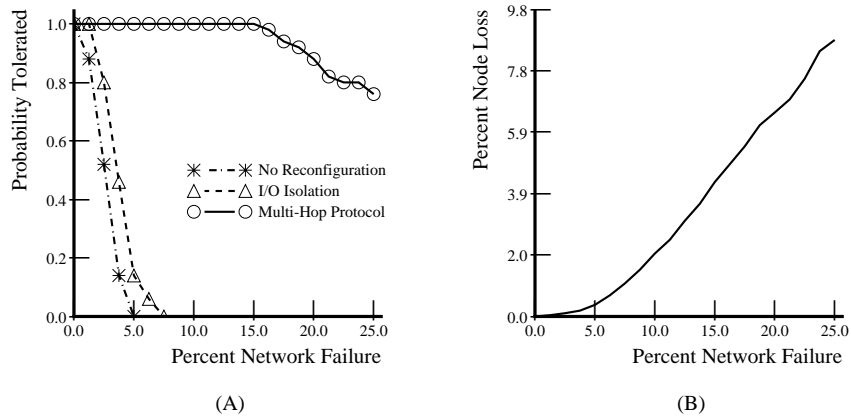


Figure (A) compares system fault tolerance given no reconfiguration, I/O-isolation, and I/O-isolation combined with the multi-hop protocol. The addition of the multi-hop protocol substantially increases fault tolerance, but is only significantly reliable for fault levels under 15 percent. Figure (B) plots the average percentage of node loss for *only those systems which tolerated failure* under the multi-hop protocol. Trials in which the system was unusable are not factored in. These trials will be included in Figure 8(A).

Figure 6: Multi-Hop System Tolerance and Node Loss for 1024-Node Multibutterfly Systems

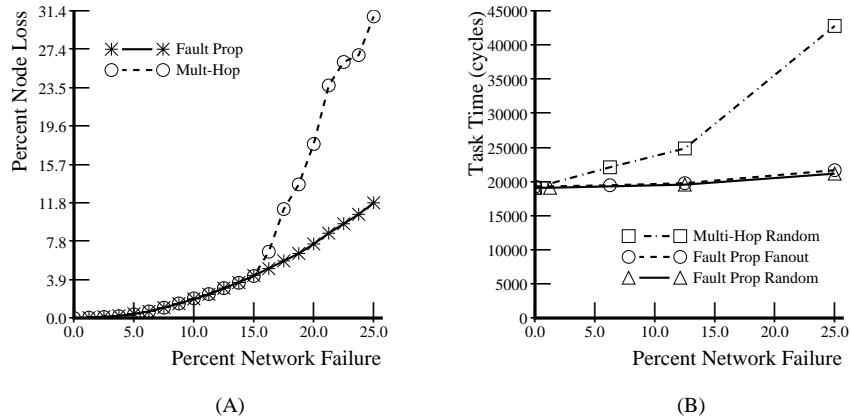


Figure (A) compares the percentage of node loss under the criterion of fault-propagation to losses under the multi-hop protocol. In contrast to Figure 6(B), the multi-hop numbers have been normalized here to *include those trials in which the system failed* (i.e. all nodes lost). Figure (B) compares the performance of the fault-propagation and the multi-hop protocol. *Fault Prop Random* refers to the performance of a randomly-wired multibutterfly using the fault-propagation algorithm. *Fault Prop Fanout* refers to a randomized, maximal-fanout network, also using fault-propagation. Somewhat surprisingly for such a small network, the multibutterfly performs just as well in practice as Theorem 2 predicts asymptotically. Also of interest, the randomized, maximal-fanout network performed as well as the multibutterfly.

Figure 8: Fault-Propagation Node Loss and Performance for 1024-Node Systems

system can be quite high.

On the other hand, we can trade some time overhead for hardware overhead by using redundant processors. We can use redundant processing nodes to shadow the computation of the processors doing the actual work [Len88] [Web90]. Actually, we need not have entirely redundant processors; we can just have redundant processes spread across all the nodes. Each node can have some “real” work and some redundant work, as long as all of the redundant processes are for work being done on other nodes. If a node is lost, all of its processes can be recovered from redundant processes. It is important to note, however, that there could be significant overhead in the messages necessary to support process shadowing. For both checkpointing and redundant processors, it is vital to place backup elements at network endpoints which are unlikely to all fail at once. For instance, backups should be placed in different clusters of endpoints in the deterministic network.

Since the rate of node loss may be quite low, it may be more appropriate to run with as little overhead as possible, and then evacuate a node’s state when absolutely necessary. Each node has two outputs and two inputs connected to the network. However, recall that the RN1 routing component is circuit switched and bidirectional. Instead of having two inputs and two outputs, each node actually has four bidirectional channels through which we can evacuate state. Depending upon the amount of state that must be evacuated, a node may choose to begin evacuation when 2 or 3 of these bidirectional channels have failed. In truly desperate circumstances, a node could even resort to the very low bandwidth of the bit-serial diagnostic network to evacuate some state.

As discussed in Section 3, boundary scan capabilities allow us to dynamically diagnose and reconfigure for both component and wire failures. How can we dynamically implement the partitioning algorithms described in Section 6?

The multi-hop protocol is fairly straightforward to implement. The Boundary Scan Controller (BSC) can look at the first and last stages of the network and calculate I/O-isolation. Multiple hops can be supported by means of a *forwarding message*, which is sent to an intermediate node and contains both the original message and its final destination. Each node can decide when to send a forwarding message in one of two ways. First, a node may decide to send to an intermediate destination after a constant number of retries to the true destination. This is the scheme used in the simulation results presented in Section 6. Second, to avoid some of the expense of multiple retries, each node may keep a cache of difficult nodes to reach, sending forwarding messages when a message’s ultimate destination matches an address in the cache. Cache entries should be replaced or flushed periodically because some nodes may be difficult to reach due to temporary congestion, not I/O-isolation. Also at issue is how to choose an intermediate node. This can be done either randomly, or with care to avoid nodes difficult to reach. Our simulations choose randomly.

On the other hand, Section 6 results emphasize the importance of implementing the fault-propagation algorithm. The algorithm requires the calculation of router blocking information.

Fortunately, the Transit architecture has hardware support for calculating this information for purposes of flow control. This flow control, however, is in use during routing while the system is in operation. We suggest three alternative methods of calculating blocking information for fault propagation. First, we can simply stop the machine and use the flow control hardware. Second, we can temporarily stop using flow control for routing, since the router will achieve reasonable performance by making random, uninformed decisions for a short time. We can use the hardware for fault propagation and then reactivate flow control. Third, if speed is not an issue, we can have the BSC gather the needed fault information and calculate fault-propagation sequentially.

Several issues have arisen in our attempt to bring multipath MIN architectures closer to reality. Unlike most previous work, we have examined systems which can tolerate node failure and isolation. We have demonstrated that a simplified version of an algorithm from Leighton and Maggs [LM92] effectively determines which nodes to use in the presence of network failure. This fault-propagation algorithm is effective not only on multibutterflies, but also on other multipath networks such as maximal-fanout networks.

Of particular importance is the quality of network connections available to each processing node. In reconfiguring a faulty system, an naive approach is to maximize the number of live nodes. However, our performance results show that the synchronization requirements of applications make it critical to eliminate poorly connected nodes from system operation. The blocked router criterion, used in the fault-propagation algorithm, eliminates such nodes and ensures high quality connections between every pair of live nodes.

Overall, we have empirically demonstrated the high fault tolerance and performance of multibutterflies for practical systems. Although lacking the theoretical basis of multibutterflies, maximal-fanout networks appear to be equally useful.

We would like to thank Tom Leighton for his suggestions on the analysis and randomization of maximal-fanout networks. Many of the architectural facets of Transit presented here are the work of André DeHon. Eran Egozy wrote the original code to generate network wirings. We would also like to thank André DeHon, Mike Ernst, Mike Klugerman, Tom Leighton, Tom Simon, Patrick Sobalvarro, Ellen Spertus, and Deborah Wallach for their helpful comments on drafts of this paper.

[ALM90] 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.

- [BP74] L. A. Bassalygo and M. S. Pinsker. Complexity of optimum nonblocking switching networks without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.
- [CED92] Frederic Chong, Eran Egozy, and André DeHon. Fault tolerance and performance of multipath multistage interconnection networks. In Thomas F. Knight Jr. and John Savage, editors, *Advanced Research in VLSI and Parallel Systems 1992*, pages 227–242. MIT Press, March 1992.
- [CFKA90] David Chaiken, Craig Fields, Kiyoshi Kurihara, and Anant Agarwal. Directory-based cache-coherence in large-scale multiprocessors. *IEEE Computer*, 23(6):41–58, June 1990.
- [Com90] IEEE Standards Committee. *IEEE Standard Test Access Port and Boundary-Scan Architecture*. IEEE, 345 East 47th Street, New York, NY 10017-2394, July 1990. IEEE Std 1149.1-1990.
- [CS82] L. Ciminiera and A. Serra. A fault-tolerant connecting network for multiprocessor systems. In *Proceedings of the 1982 International Conference on Parallel Processing*, pages 113–122. IEEE, August 1982.
- [D 92] William J. Dally et al. The message-driven processor: A multicomputer processing node with efficient mechanisms. *IEEE Micro*, pages 23–39, April 1992.
- [DeH90] André DeHon. MBTA: Modular bootstrapping transit architecture. Transit Note 17, MIT Artificial Intelligence Laboratory, April 1990.
- [DeH91a] André DeHon. Practical schemes for fat-tree network construction. In Carlo H. Séquin, editor, *Advanced Research in VLSI: International Conference 1991*. MIT Press, March 1991.
- [DeH91b] André DeHon. RN2 proposal. Transit Note 44, MIT Artificial Intelligence Laboratory, April 1991.
- [DeH92] André DeHon. Using IEEE-1149 TAP in a fault-tolerant architecture. Transit Note 60, MIT Artificial Intelligence Laboratory, January 1992.
- [DKM91] André DeHon, Thomas F. Knight Jr., and Henry Minsky. Fault-tolerant design for multistage routing networks. In *International Symposium on Shared Memory Multiprocessing*. Information Processing Society of Japan, April 1991.
- [JA92] Kirk Johnson and Anant Agarwal. The impact of communication locality on large-scale multiprocessor performance. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, Queensland, Australia, May 1992. To appear.
- [Lei85] Charles E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.
- [Len88] Daniel E. Lenoski. A highly integrated, fault-tolerant minicomputer: The nonstop CLX. In *Digest of Papers—Compcon Spring 88: Intellectual Leverage*, pages 515–519, San Francisco, California, February 1988. IEEE.
- [LLM90] Tom Leighton, Derek Lisinski, and Bruce Maggs. Empirical evaluation of randomly-wired multistage networks. In *International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 1990.
- [LM89] Tom Leighton and Bruce Maggs. Expanders might be practical: fast algorithms for routing around faults on multibutterflies. In *IEEE 30th Annual Symposium on Foundations of Computer Science*, 1989.
- [LM92] Tom Leighton and Bruce Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computing*, 41(5):1–10, May 1992.
- [MDK91] Henry Minsky, André DeHon, and Thomas F. Knight Jr. RN1: Low-latency, dilated, crossbar router. In *Hot Chips Symposium III*, 1991.
- [RK84] S.M. Reddy and V.P. Kumar. On fault-tolerant multistage interconnection networks. In *Proceedings of the 1984 International Conference on Parallel Processing*, pages 155–165. IEEE, August 1984.
- [SW89] A. Prasad Sistla and Jennifer L. Welch. Efficient distributed recovery using message logging. In *Proceedings of the Eighth Annual Symposium on Principles of Distributed Computing*, pages 223–238, Edmonton, Alberta, Canada, August 1989.
- [SY85] Robert E. Strom and Shaula Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, 3(3):204–226, August 1985.
- [Thi91] Thinking Machines Corporation, Cambridge, MA. *CM5 Technical Summary*, October 1991.
- [Upf89] E. Upfal. An deterministic packet routing scheme. In *21st Annual ACM Symposium on Theory of Computing*, pages 241–250. ACM, May 1989.
- [VR88] A. Varma and B.D. Rathi. A fault-tolerant routing scheme for unique-path multistage interconnection networks. Technical Report IBM RC 13441, IBM Research Center, Yorktown Heights, NY, January 1988.
- [Web90] Steve Webber. The Stratus architecture. Stratus Technical Report TR-1, Status Computer, Inc., 55 Fairbanks Blvd., Marlboro, Massachusetts 01752, 1990.