International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

20 Fault-Tolerant Design for Multistage Routing Networks

André DeHon, Thomas Knight Jr., and Henry Minsky

As the size of digital systems increases, the average length of time between single component failures diminishes. To avoid component related failures, large computers must be fault-tolerant; that is, the computer must perform correctly even when some components fail. In this paper, we concentrate on providing fault tolerance in the interconnection network for massively parallel MIMD computers. Particularly, we focus on methods for achieving a high degree of fault tolerance in multistage routing networks. We describe a multipath scheme for providing end-to-end fault tolerance on large networks. The scheme improves routing performance while keeping network latency low. We also describe the novel routing component, RN1, which implements this scheme, showing how it can be the basic building block for fault-tolerant multistage routing networks.

20.1 Introduction

As we begin to design and construct larger computers, the issue of fault tolerance becomes increasingly important. The number of components increases with the size of the system. As the number of components increases, the length of time between single component failures necessarily decreases. If the system is incapable of operating correctly when components fail, the mean time to failures (MTTF) for the entire system decreases similarly. In the extreme case of very large systems, the MTTF becomes intolerably small. Even in moderately sized systems, this decrease in MTTF increases the frequency of downtime and the need for repair.

Any system which hopes to acheive some measure of scalability must be designed to offset the inevitable decrease of MTTF that accompanies increasing system size. While technology and processing improvements will have some affect on the achievable MTTF for single components, these improvements will not occur at a sufficient pace for us to rely on them to keep the system MTTF at an acceptable level. We are thus forced to seek other means to offset the impending increase in the failure rate of the system.

Given that the MTTF for any component is essentially constant, we wish to improve the system MTTF. This can effectively be done by designing the system so that it can operate when some of the components in the system are disfunctional. Multiple component failures must then accumulate in order for the system to be rendered inoperative. The more faulty components the system can tolerate simultaneously, the longer the MTTF.

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

Certainly, system failures are least tolerable when they are unanticipated. The effect of component failures can be further ameliorated when the system is capable of tolerating many faults and identifying the faults as they occur. Knowing which components have failed allows the failures to be repaired before the system is rendered inoperative. The downtime for component repair can be scheduled and will thus be less costly and inconvenient than are sudden and unexpected system failures.

In this paper, we describe a scheme for achieving a reasonable level of fault tolerance in the network of a massively parallel MIMD computer by providing multiple paths through the network between each pair of network endpoints. Connections are arranged so that any of several distinct routing components at each stage of routing can be used to route to the desired destination. We do not concern ourselves with fault tolerance issues outside of the network. The design presented is applicable across the wide range of networks constructed using multiple routing stages, including all kinds of banyan-like networks [10] and fat-tree networks [13] [6] [3].

20.2 Background

20.2.1 Network Processor Interface

In order for the network to be useful in the context of a large-scale parallel computer, it must interface coherently with the network endpoints. For a large parallel computer, each endpoint will consist of a processor and memory. A typical network processor interface suitable for a shared-memory multiprocessor is shown in Figure 20.1. Here each network endpoint is a processor with its own local memory and a cache-controller. The cache-controller is responsible for coordinating the interactions of the network, the processor, and the local memory as well as maintaining its local cache and keeping the cache coherent with the rest of the network. The exact details of the connection between the processor and the network are a separate architectural issue. In general, the processor has some number of inputs from and some number of outputs to the network. Multiple connections to and from the network are necessary in order to prevent any single routing component or wire in the network from being critical.

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

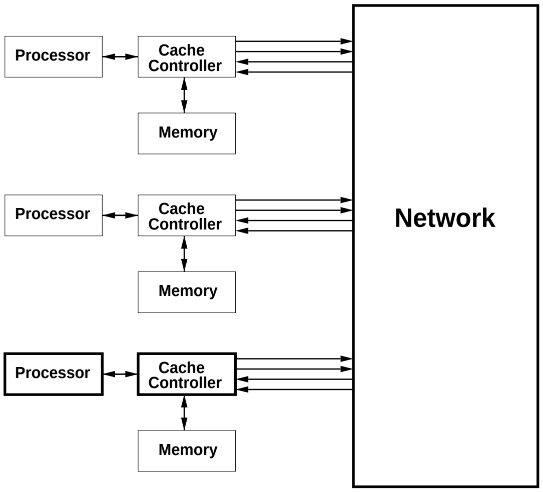


Figure 20.1
Network Processor Interface

20.2.2 Critical Components

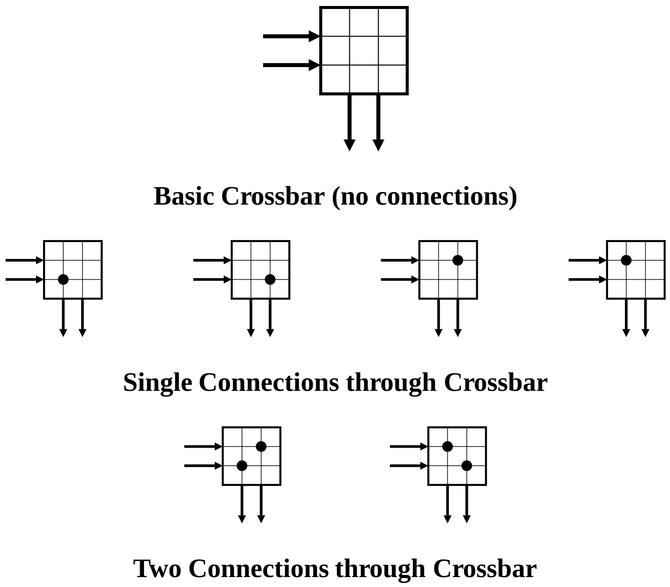
The term **critical** is used throughout this paper to refer to a component or wire when it must function properly in order for the system to operate correctly. A component is non-critical if the system can continue to operate correctly, perhaps with degraded performance, when the component fails.

In most current computer designs all components are critical. The most notable exception is the memory systems of many modern computer systems. Many computers use Error-Correcting Codes [15] [5] in their memory systems to tolerate faults in memory components. The Symbolics LISP Machine and Thinking Machine's Connection Machine are examples of computers that use ECC to protect their memory systems. The LISP Machine uses ECC on its main memory [17] while the Connection Machine uses ECC on its Data Vault disk memory [18].

20.2.3 Multistage Networks

In multistage routing networks, each routing component is effectively a small crossbar. Traditional crossbars have i inputs and r outputs and can connect any of the i inputs to any of the r outputs with the restriction that only one input can be connected to each output at the same time. Each of the r outputs is logically

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992



Basic Crossbar (no connections)

Single Connections through Crossbar

Two Connections through Crossbar

Figure 20.2
2×2 Crossbar Configurations

distinct. That is, all of the outputs route in logically different directions. If more than one input wishes to connect to the same output direction, all but one of such inputs are blocked. The outputs in each logical direction each connect to exactly one routing component in the next routing stage; this connection is made over a single physical group of wires. The number of different routing directions a routing component distinguishes, r, is often referred to as the component's **radix**.

Figures 20.2 shows the simple 2×2 crossbar routing element and its possible configurations. The input wires and output wires are orthogonal to each other. Each input and output wire runs across the chip so that any input can be connected to any output. Dots are used in Figure 20.2 to denote when two wires are connected to each other. Multiple inputs can be connected to outputs simultaneously as long as each input connects to a different output.

Figure 20.3 shows how a square network with 16 inputs and outputs can be constructed using the 2×2 crossbar routing elements shown in Figure 20.2. Input and output nodes are shown on separate sides of the network to keep the diagram simple; each pair of input and output nodes can represent a sin-

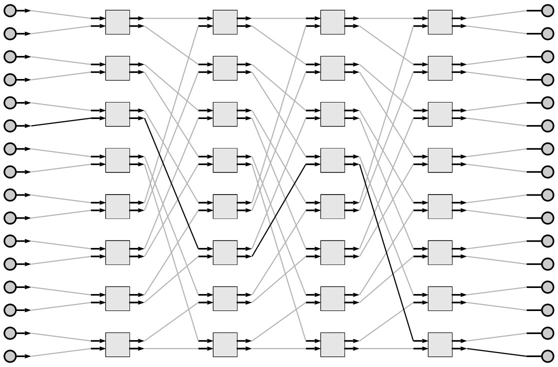


Figure 20.3
16 × 16 Square Multipath Network Constructed from 2 × 2 Crossbars

gle component. The highlighted path through the network shows the path a connection would take from processor 6 to processor 16.

20.3 Network Inputs

Each network endpoint must have multiple input connections to the network in order to prevent any single wire or routing component in the first stage of routing from being critical. Network inputs from a single endpoint should connect through many different physical components to maximize fault tolerance. For banyan-style multistage networks, all inputs to the first routing stage are **logically equivalent**. That is, connections through all inputs to the first stage routers are capable of reaching the same destination with the same routing specification. Thus, inputs from the same endpoint can easily be spread across multiple routing components. In tree structures, such as fat-trees, only small sets of inputs are logically equivalent. In order to obtain maximal fault tolerance in tree topologies, there must be at least as many components composing each set of logically equivalent inputs as there are input connections for each endpoint.

With n inputs from each endpoint, n failures can isolate an endpoint from the network in the worst-case in which all faults concentrate around a single endpoint. More than n failures can be sustained as long as no more than $(n-1)$ failures are concentrated around a single endpoint. Whether or not the complete

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

loss of an endpoint from the network is sufficient to cause the entire system to fail depends on the fault tolerance of the computational paradigm being used.

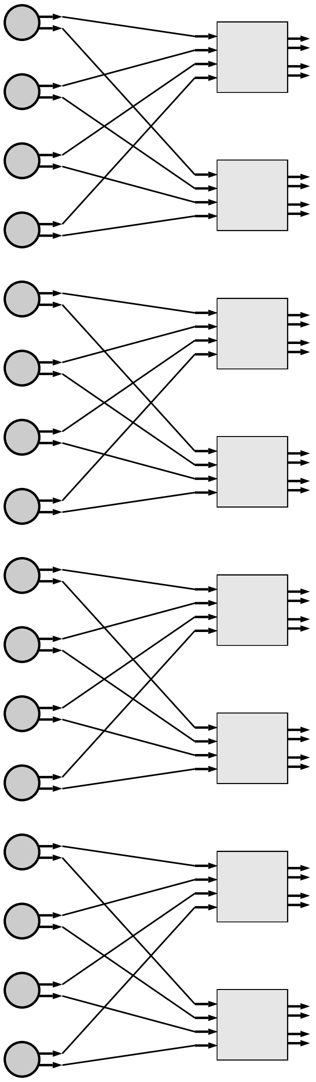
Additional wiring constraints can be utilized to minimize the effects of multiple component failures. Consider, for example, Figures 20.4 and 20.5. These two figures show the connection of processors to the first stage of routing components where each processor's inputs are attached to different physical routing components. In Figure 20.4, if the first two routing components in the first stage of routers fail, four processors are cut off from the network. However, in Figure 20.5, if the first two routing components fail, only one processor is isolated from the network. In fact with the configuration in Figure 20.5, any two component failures in the first stage of routing will isolate at most one processor from the network; similarly, any three component failures will isolate at most two processors from the network. For this simple case where each processor has two inputs to the network, the additional wiring constraint used to generate the wiring pattern in Figure 20.5 is: *no two processors sharing one first stage router should also share a second first stage router*. In a more general sense, the wiring in Figure 20.5 degrades more slowly than that of Figure 20.4 because the inputs have a greater fan-out or **expansion** into the network. A more formal characterization of expansion is provided in [12]. In contrast, it is possible to isolate one input in the expansive wiring (Figure 20.5) by any two faulty routers in the first stage; in the non-expansive wiring (Figure 20.4), two faulty routers in the first stage will only isolate an input when the routers fail as one of the four sets of pairs. In fact, it is possible to lose as many as half of the routing components in the Figure 20.4 without severing any endpoints from the network. The non-expansive case degrades less gracefully, but is less likely to degrade in a manner that isolates endpoints.

20.4 Paths Inside the Network

Considering the traditional approach to multistage networks, we see that a single faulty component or wire in the network will prevent some inputs from reaching some outputs. That is, all the components and wires involved in routing between two network endpoints are critical to the functionality of the network. This can easily be seen by reviewing Figure 20.3. Each route between a given input and output can traverse exactly one path. If a single wire or component fails, some input will be isolated from some output.

To avoid making the internal network wires and routing components critical, we must redesigned the crossbar to provide redundant paths through the

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992



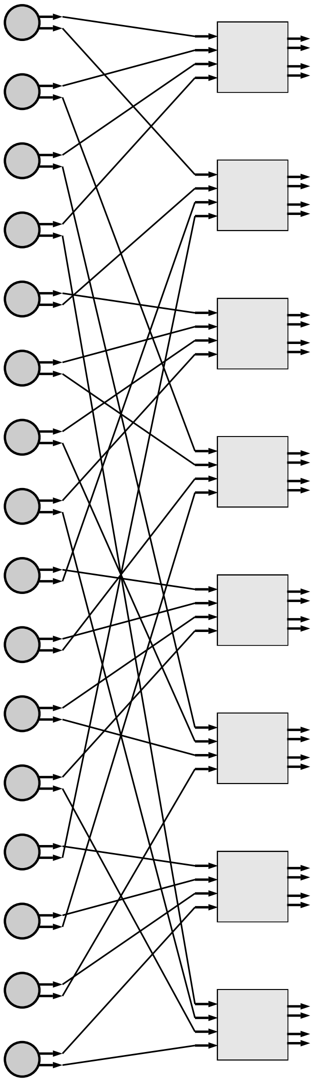
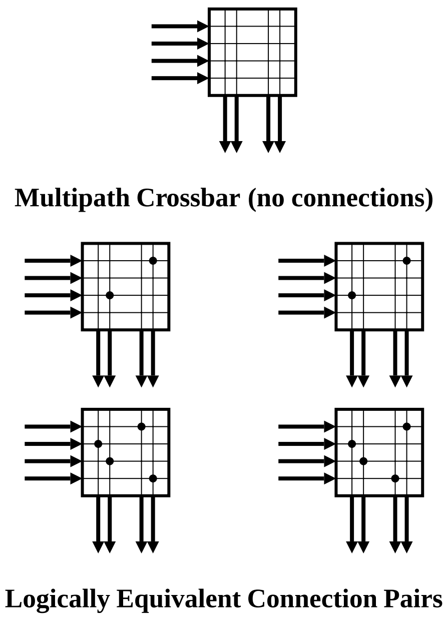


Figure 20.4
Left: Non-expansive Wiring of Processors to First Stage Routing Elements
Figure 20.5
Right: Expansive Wiring of Processors to First Stage Routing Elements



Multipath Crossbar (no connections)

Logically Equivalent Connection Pairs

Figure 20.6
4×2 Crossbar with a dilation of 2

network. We can give each crossbar element multiple logically equivalent outputs in each logical direction. Two or more outputs are considered **logically equivalent**, when they can be reached with the same routing sequence and they connect to logically equivalent inputs. A router distinguishing r logically distinct destinations with d outputs in each logical direction will have a total of $r \cdot d$ outputs. The number of logically equivalent outputs in each logical direction from a routing component, d, is referred to as the **dilation** of the router. Outputs going in the same logical direction can be connected to distinct physical routing components. The number of possible paths through the network can, up to a point, expand at each routing stage. No single wire or routing component within the network remains critical.

If we consider that any connection entering the network can start through any of n routing components in the first stage and that the number of paths increases through the network, it is easy to see that the number of input connections provides a tighter bound on the worst-case number of tolerable failures than

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

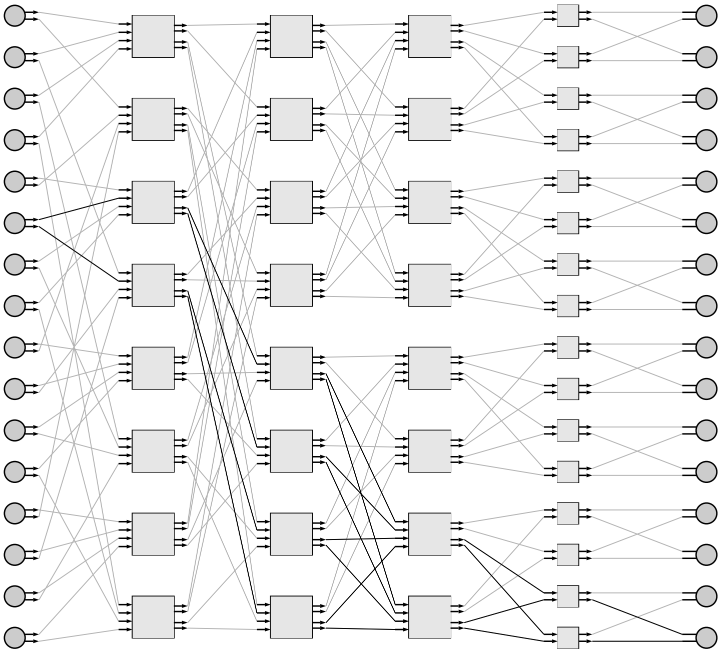


Figure 20.7
16 × 16 Square Multipath Network Constructe■ from 4 × 2 Crossbars with a Dilation of 2

one would derive considering only internal routing component failures. The same kind of consideration can be applied to the number of outputs provided to each endpoint by the network. Section 20.6 expands this reasoning to provide a quantification of the number of paths through the network.

As mentioned for the network inputs, the redundant outputs from each routing component should be connected to as many distinct physical routing components as possible to maximize fault tolerance. Expansion is just as important for connections between routers in subsequent routing stages within the network as it was for input connections. [12] characterizes this notion of expansion.

Figure 20.6 shows a 4 × 2 crossbar routing component with 2 outputs in each logical direction. Up to two inputs can be simultaneously routed in each logical output direction. Figure 20.7 shows a 16 × 16 multipath network constructed from the redundant output 4 × 2 crossbar routers shown in Figure 20.6. For comparison with Figure 20.3, all the wires which could be used to route a

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

connection between processor 6 and processor 16 are highlighted. Figure 20.7 illustrates that there are always multiple links between routing stages which can make the connection; additionally, there are multiple routing components at each stage that could be used to make the connection.

Redundant paths through the network also improve network routing performance by reducing the probability that connections will block each other within the network. [7] shows the effects of these multiple paths on network routing statistics for a specific configuration. Knight and Sobalvarro describe tools for making more general performance comparisons in [8].

20.5 Network Outputs

As is the case with network inputs, there must be multiple output connections from the network to each endpoint. Multiple output connections prevent any single wire or routing component from being critical.

Using the crossbar routing component described in the previous section, each routing component would supply multiple outputs to each endpoint. From a fault tolerance perspective, this is non-optimal since this means a single component failure would sever multiple outputs to a single endpoint. If each endpoint had m output connections and a crossbar router with a dilation of d were used, an endpoint could be isolated from the network by only $\frac{m}{d}$ faults. To maximize the number of tolerable faults for a given number of output connections (m), the dilation (d) must be minimized. At the final routing stage, then, fault tolerance is maximized by using crossbar routing elements with a single output in each logical direction.

Using crossbar routers with a single output in each logical direction in the final stage of the routing network will give the network slightly inferior routing performance to a similar configuration in which crossbar routers with multiple outputs per logical direction are used in the final routing stage. However, the improvement in fault tolerance is considerable and generally worth the tradeoff.

Note that the last stage of the network in Figure 20.7 was constructed using standard crossbar routing components like the ones shown in Figure 20.2. Using these crossbars with a dilation of one, two separate routing components can provide an output to each endpoint. If the 4×2 crossbars with a dilation of 2 had been used in the final stage, then a single component would be providing outputs to each network endpoint; this single component would then be critical for the network to be fully operational.

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

20.6 Total Path Expansion

To quantify the fault-tolerant properties of these networks we can count the number of components in each routing stage which are available to route a connection between any pair of endpoints. In the previous sections five parameters have been used to characterize the multistage network: the number of input connections per endpoint (n), the number of outputs to each endpoint (m), the number of inputs to each crossbar router (i), the switch radix (r), and switch dilation (d). Assuming all the routing components are identical, we can roughly quantify the number of paths through the network. Let N be the number of routing stages in the network. The number of paths between a single source destination pair expands further away from the source into the network at the rate of dilation, d. Thus, we have $p_{in}(s)$, the number of paths to stage s given by Equation 20.1.

$$p_{in}(s) = n \times d^{[s-1]} \tag{20.1}$$

After a point in the network, the paths will have to diminish in order to connect to the proper destination. Looking backward from the destination node, we see that the paths must grow as the network radix, r. This constraint is expressed in Equation 20.2.

$$p_{out}(s) = m \times r^{[(N+1)-s]} \tag{20.2}$$

These two expansions must, of course, meet at some point inside the network. This occurs when p_{in} and p_{out} are equal. We may call this turning point stage s'. s' can be determined as follows:

$$\begin{aligned} p_{out}(s') &= p_{in}(s') \\ n \times d^{[s'-1]} &= m \times r^{[(N+1)-s']} \\ s' &= \frac{(N+1)\cdot \ln(r) + \ln(m) + \ln(d) - \ln(n)}{\ln(r) + \ln(d)} \\ s' &= \frac{(N+1)\cdot \ln(r) + \ln\left(\frac{m\cdot d}{n}\right)}{\ln(r\cdot d)} \qquad (20.3) \end{aligned}$$

Once Equation 20.3 is solved for s', we can quantify the number of connections into each stage of the network by Equation 20.4.

$$p(s) = \begin{cases} n \times d^{[s-1]} & s < s' \\ \min(d^{[s-1]}, r^{[(N+1)-s]}) & s = s' \\ m \times r^{[(N+1)-s]} & s > s' \end{cases} \tag{20.4}$$

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

s	1	2	3	4	5
$p(s)$	2	4	8	4	2

Table 20.1
Connections into Each Stage

Note that Equation 20.4 expresses the maximum achievable number of paths between stages for a single source-destination pair. Not all wiring patterns will actually achieve this maximum between every pair of sources and destinations. In any case, Equation 20.4 provides a good first-order estimate of the number of paths available. The total number of distinct paths between each source and destination simply grows as Equation 20.1 and is thus given by Equation 20.5.

$$p_{total}(s) = n \times d^{[N-1]} \tag{20.5}$$

For the sake of example, consider the network in Figure 20.7 ($m = n = d = r = 2$, $i = 4$, $N = 4$). Solving Equation 20.3 for s', we find $s' = 3$. The number of connections into each stage can then be calculated as shown in Table 20.1. The total number of paths is simply $2 \times 2^3 = 16$. Noting Figure 20.7, we see it does achieve this maximum path expansion for the highlighted path; the paths between all other source and destination pairs in Figure 20.7 also achieve this path expansion.

20.7 Parameter Freedom

In the previous section we identified a number of parameters which characterize multistage networks (n, m, i, r, d). Additionally, the network is characterized by the number of endpoints it supports (N_p). While these parameters have been discussed separately, they are certainly not free to be specified completely independent of one another. The bandwidth into the network from the endpoints must match the bandwidth into the first stage of routing. The bandwidth between network stages must match. The bandwidth out of the network must match the output bandwidth to the endpoints. The number of processors is usually a power of the radix of the crossbar routers.

Square networks (*i.e.*, $n = m$) are often good configurations [10], especially when all endpoints are being treated equally. Square networks are usually constructed from square crossbar routing elements (*i.e.*, $i = r \times d$). Bandwidth matching is moderately easy in these cases. Rectangular networks with $n < m$ are often desirable because they offer less network congestion, since the number

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

of inputs is less than the number of outputs. However, for a given network structure, the smaller the number of inputs to the network from each endpoint (n), the less fault-tolerant the network. A square network (*i.e.*, one in which the total number of inputs and outputs are equal) can gain the same advantages as the rectangular network, by only utilizing a fraction of the inputs at a given time. The network has the improved fault tolerance of the square network with the decreased congestion of a rectangular network.

20.8 Path Selection

Once we have constructed a network with redundant paths as described, there still remains the issue of how these paths are utilized. Standard multistage networks (*e.g.*, the network of Figures 20.3) have the general advantage that they are self-routing. That is, messages can be routed from source to destination using only a few bits of data from the message stream to perform routing at each stage in the network. Switching and arbitration to set up paths through the network can occur asynchronously at each routing element involved in a connection through the network without any global arbitration. It is not necessary to have global knowledge of the state of the network.

The distributed self-routing properties of multistage networks should be preserved when constructing networks with redundant paths. To achieve self-routing and fault tolerance in multipath networks, we use a circuit-switched, source-responsible, random-oblivious routing scheme.

20.8.1 Source-Responsible Protocol

Since the network can have faulty components while remaining functional, we must provide a mechanism for establishing when a connection succeeds in traversing the network. Likewise, when more connections need to be routed to a given logical output direction of a routing component than there are outputs in that logical direction, connections must be dropped due to the lack of available resources; this blocking case must also be detected. To deal with both these cases where a message can be *lost* in the network, we use a source-responsible protocol and provide a mechanism to obtain connection status. After a message has been sent, each routing component reports back to the source the outcome of its attempt to transmit the message. If all the routing components and the destination report that the connection was made as requested, the source knows that the complete connection through the network succeeded. When one of the routing components reports that it dropped the message or when a routing

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

component fails to respond properly, the source knows that the connection failed and must be retried.

20.8.2 Random Oblivious Routing

At each switching stage, one of three things can happen. In the case in which there is exactly one output connection available in the desired logical output direction, the connection will obviously get routed through the available output. In the case where no outputs in the desired logical output direction are available, the connection must be dropped. When more than one output in the requested direction are available, the routing component *randomly* selects which output to use. Thus, all connections which can be made through a given component are made.

The routing component itself does not know the location of any faults in the network and so cannot route to avoid them; instead, the routing component routes obliviously to a logically correct output. If a connection through the network fails due to congestion or faulty components, the source will know of the failure and attempt to make the connection again. Since the choice of output ports is random at each routing stage, it is likely that subsequent connections through the network will take different paths. With this random routing, it is generally possible to get a complete connection through the network in a small number of attempts even when the network has multiple faults. Considering a network with two inputs from each endpoint, in the worst case one of the routing components in the first stage fails. On average, two attempts will be required to complete a connection through the network. Since the number of components available for a connection expands towards the middle of the network, it will generally require even less retries to avoid faulty components further inside the network.

20.9 Fault Identification

Fault localization in the network is facilitated by the connection status returned by each routing component. The data returned can include a checksum on the data sent through the routing component as well as an indication of which of the outputs, if any, was actually used in routing the connection. With a knowledge of the logical direction in which the connection was destined, the

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

actual output port utilized at each stage of routing, and the point in the network where the connection was lost, the fault can be localized to the connection between exactly two components in the network. The fault can then lie in either component or in the wire connecting them. Information from additional failures can be used to further localize the fault as necessary.

Each endpoint only has connection information from its own network transactions. This necessarily means each endpoint has only a limited amount of information about faults in the network. A higher-level protocol should be used to monitor the global network state so that repairs can be scheduled as necessary.

20.10 Other Fault-Tolerant Multistage Networks

The approach predominantly used to achieve fault tolerance in multistage routing networks is to construct a network with more switching stages than are actually required to uniquely specify the destination ([11], [2], [16], [1] *et. al.*). The set of destination specifications that reach the same physical destination defines a class of equivalent paths. Since any of several paths can reach the destination, it is possible to choose a path which avoids any single fault in the network. Most of these schemes require the processor to choose its own path through the network. These schemes almost exclusively assume each endpoint has a single input and a single output connection to the network. BBN's large Butterfly Plus computers actually implement this extra stage approach to fault tolerance.

An alternative approach for fault tolerance is to simply provide multiple redundant networks ([4], [9]). The endpoint chooses a network over which to make the connection. The networks route the connections independently and reconverge at the destination endpoint. This approach does provide multiple input and output connections. Again, the endpoints are responsible for choosing fault-free paths.

Kruskal and Snir also propose a network with redundant paths using switches similar to ours in [9]. They, however, do not develop any of the details of the network. They suggest the redundant outputs from a routing element all go to the same physical component; this, of course, undermines many of the benefits of the dilated network.

Leighton and Maggs [12] suggest a related multipath network. Their theoretical work was influential to our design. In contrast to ours, their work details packet-switched routing of data presented synchronously into the network.

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

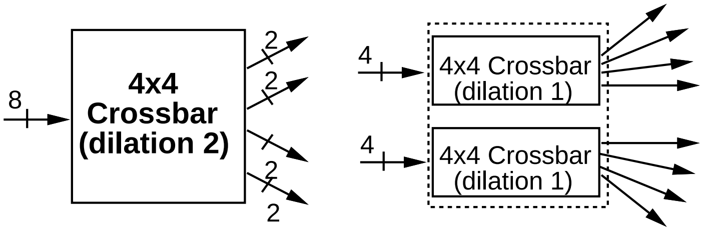


Figure 20.8
RN1 Logical Configurations

They use a much more complicated routing scheme which requires approximately $4\log_r(N_p)$ (where r is the switch radix) steps in order to route a single connection and an intricate routing switch. Our network routes in $\log_r(N_p)$ steps, but does so using oblivious routing. With the additional arbitration in their network, Leighton and Maggs can guarantee that they can simultaneously route the maximum number of packets allowed by the network's physical topology. Our network simply relies on the probabilistic properties of the data and network in order to route a large portion of connections simultaneously.

20.11 Fault-Tolerant Crossbar Routing Component: RN1

RN1 is a custom CMOS routing component constructed to provide simple high-speed switching for fault-tolerant networks. RN1 has eight nine-bit wide input channels and eight nine-bit wide output channels. These nine-bit wide channels provide byte wide data transfer with the ninth bit serving as a signal for the beginning and end of transmissions. RN1 can be configured in either of two ways, as shown in Figure 20.8. The primary configuration is a 8×4 crossbar router with a dilation of two. In this configuration, all 8 input channels are logically equivalent. Alternately, the component can be configured as a pair of 4×4 crossbars, each with 4 logically equivalent inputs and a dilation of one.

Simple routing is performed by using the first two bits of a transmission to indicate the the desired output destination. If an output in the desired direction is available, the data transmission is routed to one such output. Otherwise, the data is ignored. In either case, when the transmission completes, the RN1 routing component informs the sender of the connection status so that the sender will know whether or not it is necessary to retry the transmission. When both

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

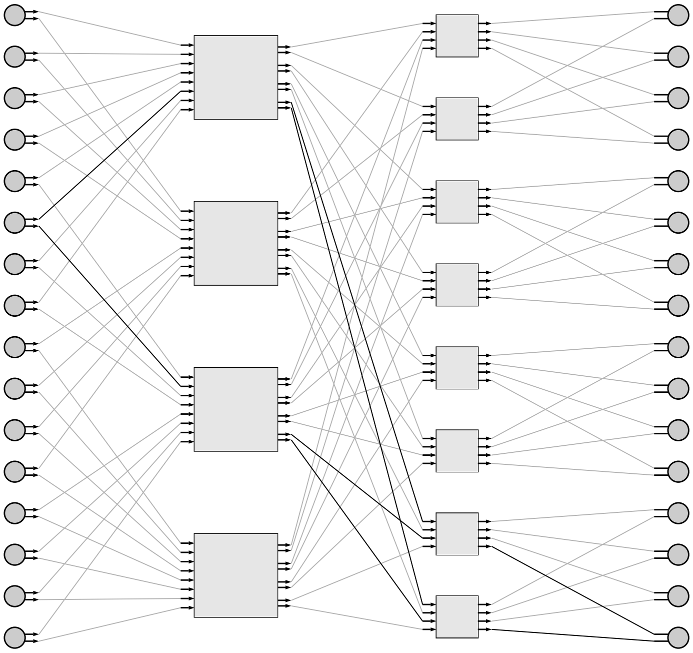


Figure 20.9
16 × 16 Square Multipath Network Constructe■ from RN1 Routing Components

outputs in the desired output direction are available, the component randomly chooses which port to use.

Figure 20.9 shows a 16 × 16 multipath network constructed from the RN1 routing component. A single physical RN1 routing component would implement two of the 4 × 4 crossbars in the second and final routing stage. To achieve the desired fault tolerance, each of the 4 × 4 crossbars in a single RN1 package should be connected to a different set of four network endpoints. As with Figures 20.3 and 20.7, the wires available for routing a connection from processor 6 to processor 16 are highlighted in Figure 20.9.

The RN1 routing component is described further in [7] and [14].

20.12 Conclusions

A high degree of fault tolerance is essential in order to build functional massively parallel computer systems. Fault tolerance can be achieved in the interconnection network by providing multiple paths through the network and multiple input and output connections to the network endpoints. Multiple paths can be realized utilizing crossbar routing components which provide multiple connections in each logical output direction. In the multipath scheme, paths through the network can be selected in a simple self-routing manner allowing cheap, low-latency interconnection. Multipath routing has the advantageous side-effect of improving the routing performance of the network. Fault recovery is facilitated by a simple source-responsible connection protocol utilizing connection status information from routing components in the network. Faults and blocking within the network can be handled in a uniform manner. The RN1 routing component implements this fault-tolerant scheme and forms the basis for fault-tolerant multistage networks.

Acknowledgments

Tom Leighton offered valuable suggestions which helped develop schemes for avoiding critical components at the network inputs and outputs. Thanks to Andy Berlin, Mike Bolotski, Max Hailperin, Pat Sobalvarro, and David Yu for reading and proofing drafts of this paper.

This research is supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

References

[1] BBN Advanced Computers Inc., Cambridge, MA. *Inside the Butterfly Plus*, October 1987.

[2] Chin Chi-Yuan and Kai Hwang. Connection principles for multipath packet switching networks. In *10th Annual Symposium on Computer Architecture*, pages 99–108, 1984.

[3] 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.

[4] Peter Franaszek and Christos Georgiou. Multipath hierarchies in interconnection networks. *Lecture Notes in Computer Science*, 297:112–123, December 1988.

[5] C. Clark Jr. George and J. B. Cain. *Error-Correction Coding for Digital Communications*. Plenum Press, New York, 1982.

[6] Ronald I. Greenberg and Charles E. Leiserson. Randomized routing on fat-trees. In *IEEE 26th Annual Symposium on the Foundations of Computer Science*. IEEE, November 1985.

International Symposium on Shared Memory Multiprocessing, 1991
Shared Memory Mulitprocessing, Edited by Norihisa Suzuki, MIT Press 1992

[7] Thomas F. Knight Jr. Technologies for low-latency interconnection switches. In *Symposium on parallel architecures and algorithms.* ACM, June 1989.

[8] Thomas F. Knight Jr. and Patrick G. Sobalvarro. Routing statistics for unqueued banyan networks. AI memo 1101, MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, September 1990.

[9] Clyde P. Kruskal and Marc Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE transactions on Computers*, C-32(12):1091–1098, December 1983.

[10] Clyde P. Kruskal and Marc Snir. A unified theory of interconnection network structure. In *Theoretical Computer Science*, pages 75–94, 1986.

[11] Duncan Lawrie and Krishnan Padmanabhan. A class of redundant path multistage interconnection networks. *IEEE Tr. on Computers*, C-32(12):1099–1108, December 1983.

[12] Tom Leighton and Bruce Maggs. Expanders might be practical: fast algorithms for routing around faults on multibutterflies. In *30th Annual Symposium on Foundations of Computer Science.* IEEE, 1989.

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

[14] Henry Q. Minsky. A parallel crossbar routing chip for a shared memory multiprocessor. Master's thesis, MIT, 545 Technology Sq., Cambridge, MA 02139, January 1991.

[15] W. Wesley Peterson and E.J. Weldon Jr. *Error-Correcting Codes.* MIT Press, Cambridge, MA, 1972.

[16] Howard Siegel. *Interconnection Networks for Large-Scale Parallel Processing.* Lexington Books, Lexington, MA, 1985.

[17] Symbolics, Inc., Cambridge, MA. *Symbolics 3600 Technical Summary*, 1987.

[18] Thinking Machines Corporation, Cambridge, MA. *Data Vault Release Notes*, version 5.0 edition, September 1988.