In *Proceedings of the 1994 Parallel Computer Routing and Communication Workshop*, May 1994, pp. 226–240.

Design of a Router for Fault-Tolerant Networks

Kevin Bolding and William Yost *

Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195

Abstract. As interconnection networks grow larger and larger, the need for reliable message delivery in the presence of faults grows as well. Unfortunately, most network routing schemes currently in use do not provide graceful tolerance of even the most common faults. Because routing messages around failed components requires non-minimal routing, it makes sense to examine routers which, by design, allow packets to take non-minimal routes. Such routers provide a basic level of fault-tolerance by allowing messages to be routed around faults, without requiring a priori knowledge of their locations. However, the mechanisms can be slow and clumsy at times. We augment Chaotic routing, a non-minimal adaptive routing scheme, with a limited amount of hardware to support fault detection, identification, and reconfiguration so that the network can automatically reconfigure itself when faults occur. We present a high-level design of these mechanisms, driven by the goal of achieving reasonable reliability without exorbitant cost.

1 Introduction

As multicomputers grow larger, the need for fault-tolerant interconnection networks grows as well. Because of the sheer number of components and links in a large multicomputer with thousands of processing nodes, the likelihood of a single failure in the network becomes too large to ignore. For instance, a 64×64-node torus network contains 16,384 links, and the same size hypercube contains 49,152 links. Since these links are often implemented as cables with plug-in connections to PC-boards, they are especially vulnerable to faults. Thus, any interconnection network designed for future large systems must incorporate mechanisms for coping with the relatively large number of faults that will occur.

Most multicomputer networks have built-in redundancy in the form of multiple possible paths from the source to the destination of a message, so it is possible for the network routers to pick paths which avoid faulty network links. Unfortunately, this task is not easily achieved for two reasons. First, routers must make their decisions very quickly, so complicated algorithms cannot be implemented. Second, global information on the presence of faults in the network is

* This work is supported in part by Office of Naval Research grant N00014–91–J–1007 and National Science Foundation grant MIP9213469.

not available, so decisions must be based on local information. Because of these constraints, the goal of building a fault-tolerant network router has been difficult to achieve.

Non-minimal adaptive routing has shown promise for providing simple, on-line mechanisms for fault avoidance [15, 2]. However, the fault-tolerance provided by such systems is not entirely robust, and can benefit from supplemental hardware that targets fault-tolerance capabilities. We present hardware and software mechanisms which boost the ability of the Chaos router to avoid faults. These mechanisms are designed to be inobtrusive and not interfere with the ordinary routing capabilities and performance of the routing network. Although the design presented here is specific to Chaotic routing, many elements of the design are applicable to other routing networks, both minimal and non-minimal, as well.

2 Fault Tolerant Routing

We define "fault tolerant routing" as a method of providing reliable message delivery between any nodes in the network connected by a functional physical path. Interconnection networks vary in the number of redundant paths they provide between nodes. However, in any network, multiple failures may render communication between two nodes impossible regardless of the routing algorithm.

The goal of fault tolerant routing has three parts:

1. Provide reliable message delivery between all pairs of nodes that have some non-faulty path between them.
2. Detect when a fault has occurred, and alert the operating system.
3. Provide mechanisms to reconfigure the network around faulty areas to ensure quick and reliable message delivery to all non-faulty areas of the network.

We use a high-level fault model in this work. Three basic classes of faults are considered: link failures, node failures and router failures, corresponding to each of the components listed in the basic node and router model shown in Figure 1. An error occurs when a packet injected into the network is either not delivered to its destination, or is delivered with corrupted data. Such faults may be either transient in nature, with only temporary effects, or persistent, causing repeatable errors. Fault detection relies on discovering that packets are lost or corrupted. Thus, if a fault occurs which causes neither lost nor corrupted packets, it may not be detected by the mechanisms introduced in this work. Soft errors such as transmission noise affecting the reception of a packet across a link or the loss of state within a router are covered by our model and are detected by the same mechanisms that detect lost or corrupted packets

Link failures refer to a fault in the physical network link between two routers. This class of failures includes common physical failures such as the unplugging of a routing cable or the unseating of a router chip from its socket. It also includes broken bond wires within the chip package. Link failures result in either the corruption of data or the loss of communication between two routers and are

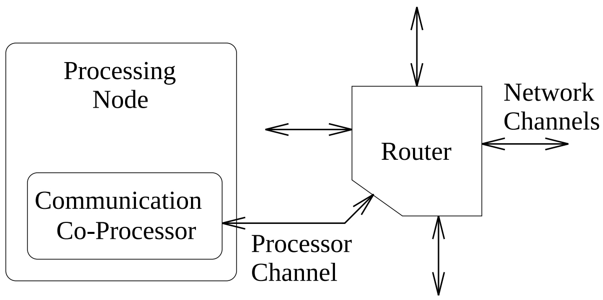


Fig. 1. A node, router, and network links. Failures may occur in any of the components or links.

easily detected. A link that is found to be faulty will be removed from service by the reconfiguration methods, but the associated routers will continue to operate.

Node failures refer to any failure which causes the processing node attached to a router to refuse to accept packets. The node appears *dead* to the network when such a fault occurs. Such faults are easily detected, although it is not possible to isolate the fault further than identifying the processing node as *faulty*. Such failures can lie either in the node itself, the communication coprocessor, or the links on the path from the node to the router. A node that is found to be faulty will be removed from service. The associated router can continue to operate so that network service is unaffected.

A router may fail in different ways, each resulting in varying loss of service. A failure in which packets are routed successfully with proper data, but use an incorrect channel or experience unnecessary delay, is difficult to detect because such behavior is within the realm of normal operation. Fortunately, such failures are not catastrophic because they do not result in corruption or loss of data. Some failures may result in certain channels of the router becoming inoperable, and will appear to be the same as link failures and be treated as such. Finally, a router may fail to operate at all. This is the most severe form of fault in the network because this renders inoperable all network links to the router and isolates the attached processing node from the rest of the network. Fortunately, router failures are expected to be more rare than link or node failures because the routers are not as vulnerable to physical effects as the physical links, and their complexity, determined by number of chips, die size, number of pins or any other measure, is much less than that of the processing node.

3 Related Work

Most current multicomputers use *oblivious* routing [7, 8, 10, 17], in which the path from a message's source to its destination is statically determined. Although such routers can be made very fast due to their simplicity, they are not robust in the presence of faults because there is no natural mechanism for avoiding faulty components. Minimal adaptive routers [4, 6, 16, 12], in which message routing has some degree of adaptivity to congestion or faults, have appeared more recently in research. Two difficulties with these schemes are that the prevention of deadlock becomes complicated and the adaptivity of message routing is limited. Recent extensions of minimal adaptive routers have added fault-tolerant capabilities, once again with the cost of added complexity [4]. Most non-minimal adaptive routers (except deflection routers) have a very natural method of fault-tolerance: faulty links appear the same as congestion, and these routers always steer messages away from congestion by *derouting* messages away from busy links [15, 14, 5]. A shortcoming of these schemes is that the natural derouting mechanisms are, in general, slow to respond when faults are present. Although deflection routers [18, 9] are quick to deroute messages, their algorithms require all links to be functional and cannot tolerate faults without special hardware considerations.

4 Chaotic Routing

Chaotic routing [14] is a form of non-minimal adaptive routing which uses randomization to prevent livelock without requiring complex protection hardware. It has been shown to be deadlock and livelock free on hypercube and many other types of networks, [1] including all k-ary d-cubes. The enhancements that we propose here are applicable to all chaotic networks. Chaotic routing provides excellent routing performance [3, 11] with a practical design [1].

Figure 2 shows a data path diagram for a two-dimensional (mesh and torus) Chaos router. A Chaos router has three main components: input/output *frames*, a *crossbar*, and the *multiqueue*. Each frame and multiqueue slot is capable of holding one fixed-size (20 16-bit words) packet. The basic operation of the Chaos router is similar to a typical oblivious cut-through packet router [13]: packets enter a router input frame, are connected through the crossbar to an output frame and have their header updated as they exit to reflect their progress. Virtual cut-through allows packets to proceed through the router as soon as their header is received and decoded; it is not necessary for the entire packet to be buffered before moving to the next router. In a Chaos router, packets may be routed to any output channel that brings them closer to their destination. Thus in this mode of operation, the Chaos router acts as a minimal adaptive router.

Chaos routers also include a central buffer, the multiqueue, which stores packets waiting for profitable channels to become free. Packets in the multiqueue have priority over packets in input frames when competing for free channels. Moreover, whenever the multiqueue becomes full, a packet is selected randomly from those in the multiqueue to be sent out the next free output channel, *regardless* of the packet's destination. By this mechanism derouting is accomplished.

4.1 Fault Tolerance

Although the design of the Chaos router was conceived with the primary goals of high performance and low complexity, a basic level of fault tolerance was a

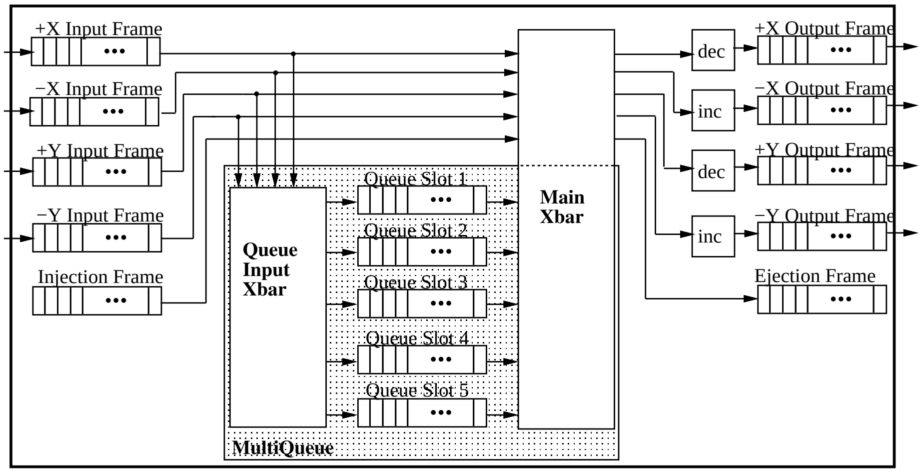


Fig. 2. Two-dimensional Chaos router data path diagram.

natural result of the design. When either a link or a router in a chaotic network persistently fails to respond to its inputs (i.e., becomes "stuck"), the output frames in neighboring routers connected to the faulty components will fail to empty, as the only way for packets to exit is through the failed component. Since packets move into an output frames only when there is space available in that frame, the routers will not route additional packets toward the faulty components.

Packets which have other profitable channels besides the faulty channel will be automatically routed out an alternative minimal path. Those which have no other profitable choices will, after a short delay, be moved into the multiqueue. When the multiqueue fills to capacity, packets will be selected on a random basis to be derouted out the next available output channel. Eventually, packets desiring failed channels will be derouted away from the profitable path and out a non-profitable, but working, channel. From its new location, the routing decision process for the packet will be reinitiated. Thus, even though the minimal paths may be blocked by faults, chaotic routing allows communication to continue by means of the derouting mechanism.

4.2 Shortcomings of the Basic Chaos Router

Although Chaotic routing provides some fault tolerance via the derouting mechanism, there is much room for improvement. First of all, packets can get permanently stuck in the output frames waiting for faulty channels. Packets already in output frames can never be derouted. Moreover, packets may be corrupted or lost due to transient faults, which do not cause the persistent "stuck-at" problems that derouting can handle.

Even when the derouting mechanism is applicable, it works only in a "clumsy" manner, for several reasons. Messages that must be derouted around faulty links may incur large delays as they wait in the multiqueue for their chance to be derouted. Moreover, under light loads, only a small number of packets may get lodged behind a faulty link. The multiqueue may never fill up enough to invoke derouting, leaving the packets stuck in the multiqueue.

Because Chaotic routing provides the promise of fault-tolerant routing, but falls short of the goal, we propose a fault-tolerant Chaos architecture to better cope with the problems caused by faulty hardware.

5 Fault Tolerant Architecture

In order to provide efficient fault detection and recovery, several enhancements must be added to the Chaos router. However, because faults are assumed to be relatively rare, only enhancements which do not significantly affect the normal, fault-free case are considered. The basic strategy for enhanced fault tolerance that we propose depends on several key points.

In order to avoid complexity and to maintain flexibility in the system with regard to procedures relating to fault tolerance, policy decisions were deferred to the software rather than hard wired into the router. For example, the router will perform some fault detection and pass that information to its associated processor, but the decision to enter a diagnostic procedure is left to the processor. In a similar manner, various watchdog timers in the router are programmable to allow the router to be tuned appropriately to maximize efficiency.

The physical context of the Chaos router limits information that it may use for diagnostic purposes to that of a local nature, specifically, internal information, or data on its connecting links. However, the actual cause of a detected problem, for example a missing packet, may be some distance away in the network. Likewise, the effects of a local fault may be felt far away. For these reasons, the Chaos router utilizes a global, synchronous procedure for fault diagnosis.

The overall network fault tolerance is accomplished mainly in system software, which orchestrates fault detection, identification, and recovery at the system level when hardware detects possible faults [2]. The initial fault detection, identification and reconfiguration is performed under the control of the local processor. Diagnostic test procedures result in the generation by each processor of a map defining its local network links as usable or unusable. On a global level, a map of the entire network is compiled from the local information provided by each processor for appropriate use by the system software.

The hardware enhancements help in the four basic phases of the fault-tolerant procedures. A special communication network is needed to allow quick *communication* for fault management. Hardware is also needed for the *detection* of faults and the *diagnosis* of their locations. Finally, once identified, mechanisms for the *reconfiguration* of faulty channels out of the system must be provided.

5.1 Communication

Communication and synchronization related to fault management are provided by a single dedicated bidirectional line per network channel that passes the *red-alert* signal. The primary purpose of this signal is to notify the entire network that a fault may have occurred so that action can be taken to affect recovery. The signal may be asserted by any node in the system, and is propagated to each of the neighbors of the asserting nodes. The router does not originate this signal but passes a signal that it receives from its associated processing node or from any other input channel. Each node that receives the red-alert signal marks the channel from which the red-alert arrived and asserts the signal on all other channels. By this means the red-alert is broadcast to all elements of the network in time proportional to the diameter of the network. Marking the arrival direction allows the node that originated the assertion of the red-alert to control its release. This facilitates control of the red-alert network by this node and allows it to use the red-alert line to signal the phases of the fault diagnosis procedure while maintaining synchronization within the network. A watchdog timer at each node ensures that a stuck red-alert line will not continually force alert status on the network.

The processor channel contains a similar line with some additional functionality. The router may strobe the *Error-detected* line in order to signal to the processing node that the router's fault detection circuitry has detected a possible problem. The processor initiates the red-alert by signalling the router via the Error-detected line. Fault detection signalled to the processor generates a trap condition which, depending upon the context and nature of the fault, may result in the initiation the red-alert by the processor.

5.2 Fault Detection

There are several mechanisms by which the Chaos router detects possible faults. Since the amount of time that a router output frame must wait until granted control of the channel is bounded, watchdog timers on each output channel can indicate that a channel is unresponsive and probably faulty. A timeout will cause the router to strobe the Error-detected line to the processor.

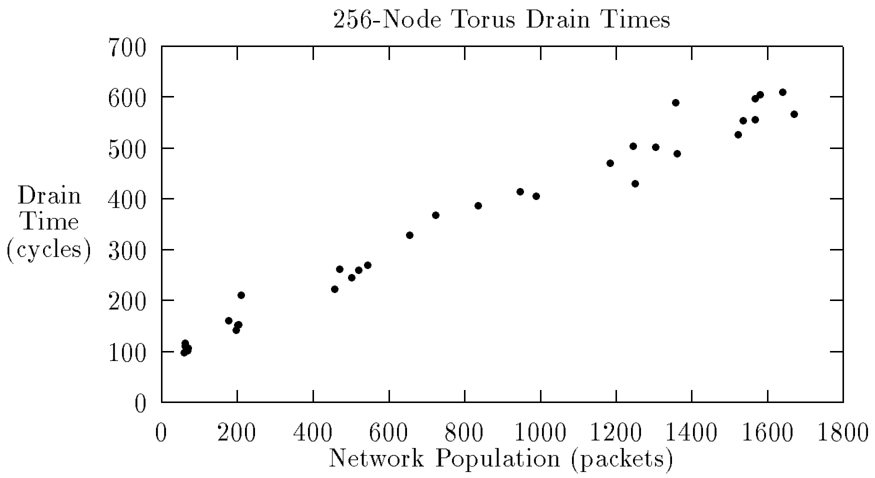
The Chaos channel protocol only requires that the tail of a packet be marked by the assertion of the End-of-message (EOM) signal. Otherwise, transfer of a word is assumed each clock cycle. A channel that failed with a packet in mid-transmission could generate a packet of potentially unbounded length. Because packet-switched routing assumes bounded length packets, we can protect against this situation by counting words for each packet transmission. If a transmission becomes larger than the maximum packet size, the router will terminate the transmission and strobe the Error-detected line.

The body of a message is protected by a checksum, which is generated and verified by the injection and delivery nodes respectively. However, the packet header contains dynamic routing information which is updated on each network hop and cannot be included in the checksum calculation. The router generates new parity for each hop and checks each incoming packet. A packet which fails the header parity check is immediately delivered to the processing node associated with the router which detects the problem, since the routing information is likely to have been corrupted. When a packet is received by the Chaos router, a map of profitable routing directions is generated which is used in making routing decisions. Parity error detection forces the map to indicate that only the delivery channel is profitable.

System Drain. Upon receipt of a red-alert, the router halts the injection of new packets into the network. This requires only that the injection channel from processor to router be disabled. A system drain results, during which packets within the network are delivered and the network empties. The drain is terminated by the expiration of a watchdog timer or when the network is empty.

Figures 3 and 4 show system drain times for 256 and 1024 node torus networks as a function of the number of packets in the network. The worst case drain time is the significant number in setting a drain timeout period. An upper bound on the time that it will take to perform a system drain in a fault-free network can be calculated by assuming a single destination for all packets. If all buffers in each router were filled with packets headed for the same destination, then the bottleneck in the network would be the single delivery channel from the destination router to the destination processing node through which all traffic would have to pass. The rate at which packets could pass on this link would determine the drain time. Given a 32×32 node network with 15 buffers per router and 20 word packets, it would take $1024 \times 20 \times 15 = 307200$ cycles to drain the network. The drain time in a faulty network may be unbounded. During the system drain phase, the red-alert signal is used to indicate that the drain is incomplete. Each router asserts the signal to indicate that it still contains undelivered packets. The release of the signal by all routers is an indication that the network has drained and the next phase of activity may be entered. If the drain has not completed after the maximum drain period, there must be packets remaining in the network that are undeliverable due to dead destination nodes or a pattern of failures that prevent the packets from reaching their destinations. This is indicated by the expiration of a timeout period within each router. In this event, remaining packets will be delivered to the local processing node . All nodes will then enter the diagnosis procedure to check for faults. The fate of those packets misdelivered at the termination of the system drain will be determined at a higher level.

The drain itself provides for some fault detection. One of the characteristics of a non-minimal adaptive router is that packets will not necessarily be delivered in order. The communication coprocessor must reorder the packets in order to reconstruct the message. As a side effect of so doing, missing packets are detected without any significant additional cost in hardware. If the buffer in which the message is reordered overflows or if the missing packet doesn't show up within a timeout period, a system drain can be initiated. If the packet is simply slow in arriving, then the drain will facilitate its delivery. If it is in fact missing, then its



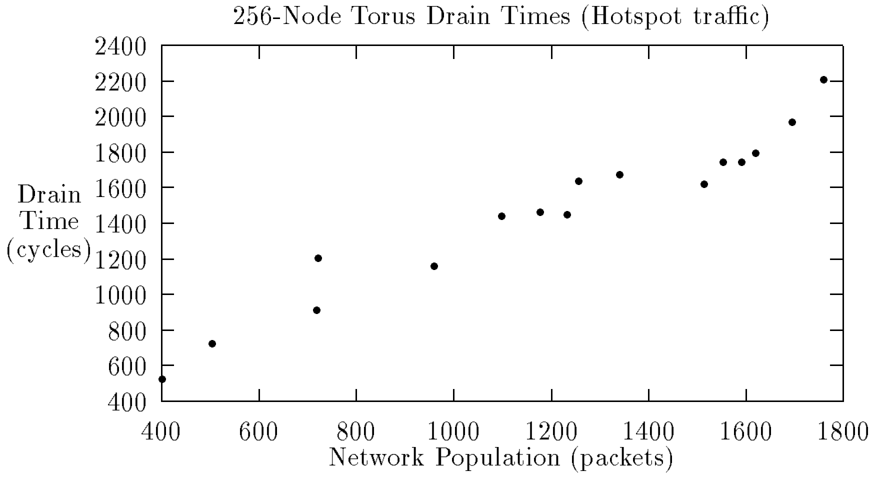
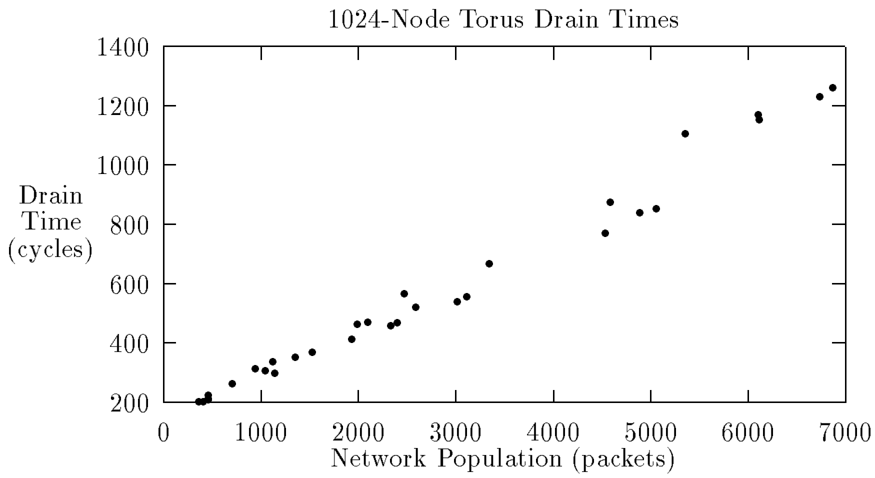


Fig. 3. System drain times for a 16×16 node torus network. Packets are injected with uniform random destinations in the top figure, and with traffic containing ten ten-times hotspots in the lower figure.

nonarrival at the end of the drain will be cause to proceed to the fault diagnosis stage.

5.3 Fault Diagnosis

Once a red-alert has been signaled and the system drained, fault diagnosis begins. The goal of diagnosis is to identify as many faults as possible by using a quick distributed algorithm. The procedure is straightforward: All nodes inject test packets destined for each of their neighbors. When a node receives a packet,



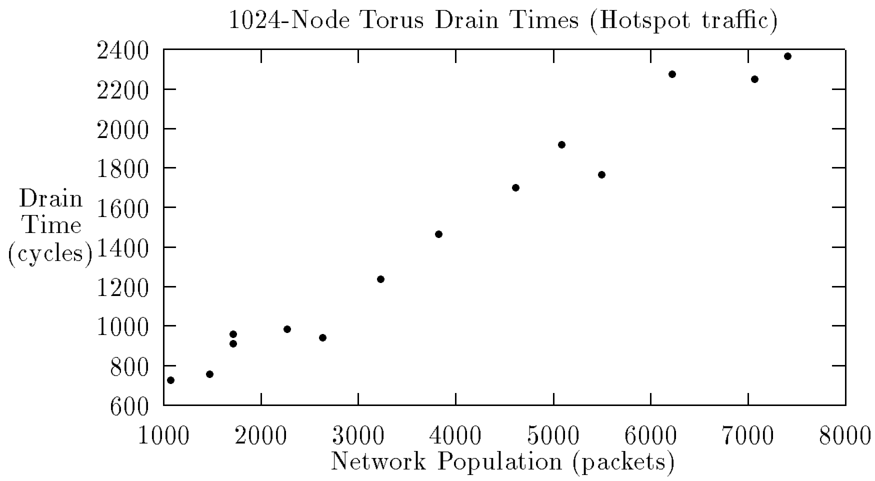


Fig. 4. System drain times for a 32×32 node torus network. Packets are injected with uniform random destinations in the top figure and with traffic containing ten ten-times hotspots in the lower figure.

it sends an acknowledgment back to the sender. This tests the path shown in Figure 5(a) in both directions. If the message is not returned, or is returned corrupted, it could be due to failure of either the link, one or both of the routers, or the receiving processing node. A node that performs a test in which it does not receive a good reply will simply flag the target link as bad. The routers and nodes will be tested by more than one test, since each node is connected to more than one link, and all links are tested. The combination of test results may provide conflicting information about a node or router's status. If this is the case, the operating system will resolve conflicts.

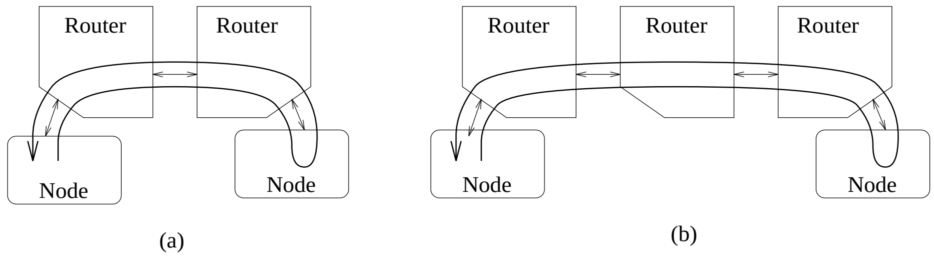


Fig. 5. Fault diagnostic testing. (a) One-hop testing path, which requires nodes to be functional for the link to pass the test. (b) Two-hop testing path allows links to pass the test even if nodes fail.

There are two problems with this test design. First, packets intended to be routed on specific links in order to test those links may be derouted around faulty links by the Chaos router and still reach their proper destination without exercising the target link. To avoid this, the router will be configured to disable derouting during the diagnosis phase. This is easily implemented in the router's hardware. The second problem is that, since the processing node must be alive to participate in the testing, a dead node will cause the tests of each of the links from its router to indicate failure. This would render a router attached to a dead or missing node useless. A way to work around this is to add a second phase of testing. In this phase, all two-hop paths would be tested, as in Figure 5(b). More hardware must be functional for this test to succeed, but the intermediate router's attached node does not play any part in the test. Thus, if a link passed the two-hop test, but failed the one-hop test, it can be inferred that the link is working, but that the node is dead. This procedure could be extended with tests of more than two hops, however, the test quickly becomes unwieldy. The number of syndromes that must be decoded based upon test results grows extremely quickly. It appears undesirable to allow extended paths through routers without active associated processors. Because the control of the network fault tolerance is embedded in the local processors, the number of such nodes with inactive processors must be limited in order to maintain reliability, We would recommend that adjacent routers with inactive processors be avoided. This is a policy decision that does not require hardware support by the router.

On-line Testing. The diagnostics described above are only run when the system has been drained and is in a special "diagnostic" state, triggered by one of the fault detection mechanisms. In order to provide more timely fault detection, diagnostics can be run on-line as well. In on-line testing, nodes inject test messages to each of their neighbors, just as above, but during normal network operation. If no response is received after some specified time period, the node can declare a red-alert and enter the off-line diagnostic phase. This solution, unfortunately, requires substantially more hardware support than off-line testing. Test messages must be identified as such so that the network interface recognize them and not attempt to pass them to the processor as normal messages. This is easily done, but requires dedicating space in the header of packets. A larger problem is the requirement that test messages not be derouted. Since on-line test packets must be sent along with ordinary operational packets, the router cannot be put in "no-deroute" mode to solve this problem. To accomplish this, test packets must be marked as such in their header, and hardware must identify these packets and force them along minimal paths. Moreover, since derouting is a critical part of the deadlock-freedom of Chaotic routing, the packets cannot be allowed to create deadlock. This can be solved with additional hardware that destroys test packets after some time limit has been reached. However, because the hardware costs and the complexity of on-line testing are quite large, it is not likely to be a practical addition to the router.

Test Coverage. The diagnostic tests described above provide simple and basic tests of the routing network. The tests will find most static failures in network links, and will detect when nodes fail enough to upset the testing protocol. Router failures are not covered fully. Because the testing path does not cover all of the possible message paths through the router, faults may escape the test. Moreover, only a rudimentary test of the router's ability to correctly route packets is made. Transient faults cannot be found through any form of standard diagnostic testing, and this scheme is no exception to that, though transient errors will be detected by our fault detection scheme. It is our conjecture that most static network failures will lie in faulty links because these require off-chip and board-to-board connections which are especially vulnerable to breakage. Thus, our diagnostics are aimed primarily at link failures and provide a low-cost and quick method of diagnosing such faults.

5.4 Reconfiguration

Once diagnosis is complete, each node has knowledge of the status of the links connected to its router. This data is used to reconfigure the router by marking faulty links as "dead." The router masks out dead channels when making routing decisions. This can be done in a straightforward manner using little extra hardware. A mask corresponding to the functional channels is created at configuration time. When new packets arrive at the router, their profitable channels are determined and logically AND'ed with the functional channel mask, so that they will avoid dead channels. If the packet's profitable channel list is zero, meaning that there are no functional profitable channels, then it must be immediately derouted. This is done by setting the profitable channel list for this packet to the functional channel mask, with the exception that the channel that the packet entered on is excluded. This prevents packets from being derouted immediately back to the source of the previous route. The hardware for this is quite simple: a logical AND, a test for zero, and a register load, so it will barely impact the critical path of packets through the router.

This reconfiguration short-circuits the normal mechanism for derouting packets around faults. Packets do not have to wait to be randomly derouted by the multiqueue to detour around faults, and packets will not get stuck behind faults, even if there is no other traffic in the network.

In order to preserve information on faults for the long term, each node must communicate fault data to a central controller. Since a single controller may be faulty, there can be a set of two or three redundant controllers, each serving the same purpose. Each controller constructs a map of all of the faulty components in the network. This information can be used for several purposes:

- Saving the global information on faults so that it can be taken into account when reconfiguring after a system re-boot.
- Communicating information on faults to the system.
- Alerting the system administrator about new faults.
- Configuring dead or unreachable nodes out of the system completely. Dead nodes should not ever be the destination of any messages.
- Removing messages destined for dead nodes.

By reconfiguring the system around faulty nodes, the network can still run efficiently despite the presence of many faulty components.

6 Summary of Hardware Costs

The total amount of additional hardware that is required to implement this proposal is a small fraction of the basic router cost in hardware. The communication network adds one line per channel and several latches and a small amount of logic for each channel. Considering that each channel must buffer 20 words of 16 bits, the incremental cost is small. The parity checking and generation takes only a few gates per channel. The output channel timers are not large, less than 10 bits and the worst case watchdog timer for any purpose is less than 20 bits. To detect when system drain has completed, each router must be able to report if it is buffering any packets. Again, the internal hardware for this is small. A small state machine is required to control the router during the system drain and the diagnostic procedures. There is also a small cost in augmenting the logic that implements the routing decisions, since the mask of dead channels must be including in that decision. Also, a number of other issues must be handled such as the delivery to the current node of packets with bad header parity as well as the delivery of undelivered packets to the current node at the termination of a system drain. These functions fit smoothly into the basic routing decision circuitry and the incremental cost is small.

7 Remaining Problems

Although our proposed design provides solutions to many problems of reliability in computer networks, some problems remain. Most of these are due to the difficulty of the problem, the solving of which requires the addition of significant complexity to the network, or the local nature of information available to our solutions. Some known shortcomings are:

- **Orphaned messages.** If a node stops reading messages destined to it from the network, the messages will back up into the network without ever being delivered. We have no way to detect such a problem except by the presence of messages in the network after a system drain times out.
- **Global problems.** While our solutions work fine for isolated faults and *convex*-shaped fault areas, certain patterns of faults can cause situations where messages get stuck in the network. To recognize such fault patterns and solve them requires global techniques.
- **Error correction.** Currently, our system only detects data errors and informs the operating system when one has occurred. Error correction, which requires more hardware and overhead, would reduce the need for applications to re-send messages and re-start applications from checkpoints.
- **Router failures.** Our system centers around data errors and link faults. Coverage of router failures is very limited. The test patterns we use for fault diagnosis do not test the multiqueues at all, and leave many paths through the router untested. Solutions that diagnose router failures will be more complicated, but may not require any additional hardware.

8 Conclusions

The large number of components in massively-parallel multicomputers makes reliability an important issue. However, most interconnection networks currently in use, having been designed exclusively for performance, do not provide even the most basic levels of fault-tolerance. Networks with non-minimal adaptive routing, such as Chaotic routing, are exceptions and provide a good basis for fault-tolerant design. By augmenting the natural fault-tolerant mechanisms with a small amount of hardware, we have designed a router that provides a higher level of fault tolerance.

The architecture of the fault-tolerant Chaos router has been designed with a goal of providing reliable service with nearly the same performance as unprotected Chaos networks. To achieve this, we have added a simple rudimentary communication scheme to assist in fault detection and diagnosis, a small number of hardware timers to check if hardware functions are operable, and some architectural modifications to the routing logic itself to provide efficient routing around network faults. The remaining functions are provided by the network interface and software, in manners which are either efficient or infrequent.

The sum of these parts is a system which provides reliable delivery of messages, as well as automatic detection of errors and diagnosis of fault locations at a reasonable cost.

Acknowledgments

We wish thank Lawrence Snyder for guidance and direction as well as many of the ideas presented in this work. Neil McKenzie's network interface design has also driven many of the issues resulting in this design.

References

1. Kevin Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router.* PhD thesis, University of Washington, Seattle, WA, July 1993.
2. Kevin Bolding and Lawrence Snyder. Overview of fault handling for the chaos router. In *Proceedings of the 1991 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 124–127, November 1991.
3. Kevin Bolding and Lawrence Snyder. Mesh and torus chaotic routing. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pages 333–347, March 1992.
4. Andrew A. Chien and Jae H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proc. Int. Symp. on Computer Architecture*, pages 268–277, May 1992.
5. Bill Coates, Al Davis, and Ken Stevens. The post office experience: Designing a large asynchronous chip. In *Proceedings of the HICSS*, 1993.
6. Robert Cypher and Luis Gravano. Adaptive, deadlock-free packet routing in torus networks with minimal storage. In *Proc. Int. Conf. on Parallel Processing*, pages 204–211, 1992.
7. W. Dally. Wire-efficient VLSI multiprocessor communication networks. In Paul Losleben, editor, *Proceedings of the Stanford Conference on Advanced Research in VLSI*, pages 391–415. MIT Press, March 1987.
8. W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. on Computers*, C-36(5):547–553, May 1987.
9. Chien Fang and Ted Szymanski. An analysis of deflection routing in multi-dimensional regular mesh networks. In *Proceedings of IEEE INFOCOM '91*, pages 859–868, April 1991.
10. C. Flaig. VLSI mesh routing systems. Master's thesis, California Institute of Technology, May 1987.
11. Melanie L. Fulgham and Lawrence Snyder. Performance of chaos and oblivious routers under non-uniform traffic. Technical Report CSE-93-06-01, University of Washington, Seattle, WA, June 1993.
12. Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *Proc. Int. Symp. on Computer Architecture*, 1992.
13. P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.
14. Smaragda Konstantinidou and Lawrence Snyder. The chaos router: A practical application of randomization in network routing. In *Proc. Symp. on Parallel Algorithms and Architectures*, pages 21–30, 1990.
15. J. Y. Ngai. *A Framework for Adaptive Routing in Multicomputer Networks.* PhD thesis, California Institute of Technology, Pasadena, CA, May 1989.
16. Gustavo D. Pifarré, Luis Gravano, Sergio A. Felperin, and Jorge L. C. Sanz. Fully-adaptive minimal deadlock-free packet routing in hypercubes, meshes and other networks. In *Proc. Symp. on Parallel Algorithms and Architectures*, pages 278–290, 1991.
17. Charles L. Seitz and Wen-King Su. A family of routing and communication chips based on the Mosaic. In *Symp. on Integrated Systems: Proc. of the 1993 Washington Conf.*, pages 320–337, 1993.
18. B. J. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proceedings of SPIE*, pages 241–248, 1981.