Cost Effective Fault Tolerance for Network Routing

by

William Yost

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

1995

Approved by____
(Chairperson of the Supervisory Committee)

Program Authorized
to Offer Degree____

Date____

Master’s Thesis

University of Washington

Abstract

Cost Effective Fault Tolerance for Network Routing

by William Yost

Chairperson of the Supervisory Committee: Professor Carl Ebeling
Department of Computer
Science and Engineering

As the size of interconnection networks for multicomputers increases, it becomes clear that some degree of fault tolerance will be required in order to maintain system reliability at an acceptable level. The interconnection networks contain large numbers of relatively unreliable components. However, cost and performance concerns mandate that the addition of fault tolerance to the network have an unobtrusive impact on network design. The target design space requires reasonable reliability while avoiding exorbitant additional costs. Because routing messages around failed components may require non-minimal routes, it makes sense to examine routers which, by design, allow packets to take non-minimal routes. However, to provide for effective and efficient fault tolerance, a unified, coherent scheme for fault management is required. Chaotic routing, a non-minimal adaptive routing scheme, is augmented with a limited amount of hardware to support fault detection, identification, and reconfiguration so that the network can maintain reliable operation in the presence of faults. The Express Broadcast Network is introduced; a low overhead control network orthogonal to the data network that can, in general, provide for fine-grained control of the network, and specifically, provides for global control and synchronization of fault management procedures for Chaos. A high-level design is presented in which fault tolerance is greatly enhanced by the addition of functionality to targeted blocks while maintaining the structure of Chaotic routing and avoiding large incremental costs.

Table of Contents

List of Figures

List of Tables

Acknowledgements

Many people have helped me in my years at the University of Washington. In particular I would like to thank Carl Ebeling who kept me on track, Larry Snyder who introduced me to chaotic routing and provided great support in the development of my research topic and Kevin Bolding whose insight into network routing provided direction to my work and whose day-to-day support kept me moving toward my goal.

Last but not least I would like to thank my family for bearing with me during this extended but rewarding process.

1 Introduction

As multicomputers grow larger, maintaining system reliability and availability at acceptable levels becomes increasingly important and difficult. Because of the sheer number of components and links in a large multicomputer with hundreds or thousands of processing nodes, the probability that a failure will occur in the system becomes too large to ignore. Even a single faulty component may generate a fanout of errors that affects the whole system. As an integral and important element of a multicomputer system, the reliability of the interconnection network makes a vital contribution to system reliability. If a design is not fault tolerant, it can only be as reliable as its weakest link, as the reliability of such a system is the product of the reliability of all system elements. Simply using highly reliable components will not provide a lasting, scalable solution. At some point as the scale of the network increases, the reliability will drop below acceptable levels. To maintain reliability as the system scales requires that fault tolerance be designed into the system.

The effect of fault tolerant design is to improve the reliability of the fault tolerant block or module beyond that of its components by maintaining functionality even in the presence of faults that cause individual components to fail. Because multiple failures must occur before thare is any loss of functionality, the reliability of the fault tolerant block ends up higher than the simple product of the reliability of all components. Fault tolerance that is built into the system can localize fault effects and allow smooth operation to continue without disruption. With proper fault and redundancy management, a fault that directly affects only a small fraction of the resources of an interconnection network can be isolated and the integrity of the larger system maintained. Ordinary scheduled maintenance can then be used to repair failures without a cost in system availability.

1.1 Fault Tolerance

Fault tolerance encompasses a number of different functions. A fault may generate an error which is an action in variance from that which would have occurred if the system were fault-free. The fault tolerant system must *detect* the error and respond. It may *mask* the fault and then simply move on or it may engage in fault *identification*. A component identified as faulty may be *reconfigured* out of the system to deactivate the fault. The system may then incrementally lose performance or a *spare* may be available to replace the function of the faulty component. A *recovery* procedure may be necessary to restore the system state to match that of a fault-free system.

For the purposes of this study, "fault tolerant routing" is defined as a means of providing for the reliable delivery of messages within a network. The goals of fault tolerant routing are fourfold:

1. Provide reliable message delivery between all pairs of nodes that have a nonfaulty path connecting them.
2. Detect when a fault has occurred, and alert the operating system.
3. Provide mechanisms to reconfigure the network around faults to ensure continued, reliable service to all nonfaulty areas of the network.
4. Provide for recovery mechanisms that allow the computation to proceed in a reliable manner even after interruption by fault management procedures.

Cost is an important parameter for a general purpose computer. Full implementation of the goals of fault tolerant routing requires a commitment of resources that is probably not cost-effective for the ordinary user. The motivation for this study is to determine means of approaching as closely as possible the goals of fault tolerant routing while avoiding exorbitant additional cost.

1.2 General Implementation of Fault Tolerance

Fault tolerance is implemented through the use of redundant resources. Fault detection and identification require that a good value (real or implicit) be available with which to compare a block or module output. A simple parity bit appended to a data word provides a limited amount of detection capability by expanding the space of data values and implicitly splitting it into "good" and "bad" sets. More complicated parity schemes can be used to provide error correction coding (ECC) which splits the expanded space of coded data values into implicit sets of "good", "bad" and "correctable" values such that data can be recovered from a faulty code word [Tang & Chien 69]. Master/shadow checking blocks provide explicit comparisons against presumed "good" values, thereby providing fault detection. Triple modular redundancy (TMR) and N-modular redundancy (NMR) take this idea further and vote three or more module outputs to generate an output value consistent with the majority of voters. Depending upon system needs, modules that lose the vote can be permanently reconfigured from the system or simply masked from the transaction in question [Sieworek & Swarz 82].

Reconfiguration implies the existence of redundant resources in the system. It would make no sense to be able to reconfigure a system to eliminate faulty components if the resulting system is unable to fulfill its allotted tasks.

Recovery must occur after the detection of an error if the fault has not been masked and the system state has been altered in error. This process requires some variety of information redundancy. If the operands of a transaction are preserved until the transaction is reliably completed then a simple retry may suffice for recovery. For example, an addition can be repeated if the operands are still present in memory, or a network transmission can be repeated if the source maintains a copy of the message. Periodic checkpointing of a system provides for coarser grained information redundancy.

Existing fault tolerant multicomputers have used various styles of redundancy. The Vulcan architecture from IBM [Stunkel et al. 94] uses the master/shadow checking pair concept, duplicating every switch element and most elements within the processing nodes. Honeywell has modified the Intel Touchstone machine to make it suitable for space operations. Their strategy uses software implemented fault tolerance in which the task environment in a node is periodically captured and replicated in another node. Additional nodes are set aside as spares to replace failed nodes. This approach does not add redundancy so much as it limits functionality in order to redefine baseline resources as redundant elements in the system.

1.3 Network Redundancy Management

Interconnection networks for large multicomputers are generally highly redundant. When each routing node is connected to three or more links, there will usually be multiple possible paths from point to point across a network. The algorithm used for routing messages affects the manner in which redundant paths may be utilized. An algorithm that allows for the use of alternative paths through the network will by nature be fault tolerant, to some extent, as it will be able to maintain network functionality in the presence of faults.

In an oblivious routing algorithm, the path a message takes through the network is determined by the relationship between the source and destination addresses. There may be multiple possible paths but an oblivious router will choose only one, generally a minimum length path. A typical implementation of an oblivious routing algorithm is a dimension order router. Dimension order routing orders the dimensions of a network and forces messages to traverse the network in that order. A message will not be routed in a given dimension until all dimensions ordered before that dimension have been fully routed. Oblivious routers are of low complexity and are relatively simple to implement while providing reasonable performance. However, the algorithms do not provide any means by which redundant network paths can be used to bypass faulty components. A faulty network link in a dimension order router will cut communication between all nodes whose connecting path must cross that link.

A minimal adaptive routing algorithm allows one of multiple minimum length paths to be chosen based on local conditions. A minimal adaptive router will determine the set of profitable directions based upon its own location and the message destination. The message will be routed onto a profitable link or, if none are available, the message will wait until a profitable link becomes available. A minimal adaptive router will seamlessly route around a faulty link *if* an alternative profitable direction is available. If such an alternative is not available, then the message is blocked. A fault does not necessarily break communication between two nodes using the faulty link because alternate paths may exist that branch before the faulty component is reached. However, some nodes will definitely lose communication, specifically those on opposite sides of the faulty link for which the link lies in the only routable dimension.

Non-minimal adaptive routing algorithms allow the router to choose any direction to route a message. However, for the sake of performance, the algorithm will usually choose a minimal route if possible. Only if no minimal route is immediately available will a non-minimal route be selected. Even though no path is ruled out algorithmically, in reality, only minimal paths or subpaths will be selected as long as they are available. In a regular, fault-free network, congestion causing contention for links will be the only reason for a packet to be routed along a non-minimal path. The non-minimal path will eventually move the packet clear of the congested area from which point it can then proceed along a minimum length path.

In a faulty network, an unresponsive link or node appears as congestion that *never* clears enough to allow a message to pass. This can trigger a non-minimal routing decision that is made without any explicit knowledge of faults but is based only upon a tabulation of the available links.

Non-minimal adaptive routers have a clear advantage over minimal and oblivious routers because of their flexibility in redundancy management that will extend the usability of the network in the presence of faults. However, they cannot handle all faulty situations. As will be shown, faults can create obstacles of significant extent that will be difficult or impossible to handle, even for a non-minimal adaptive router with enhancements for fault tolerance. Because of the need for high performance and scalability and due to the difficulty in gathering timely information from beyond the immediate neighborhood, non-minimal adaptive routing algorithms generally use local information in making routing decisions. Livelock becomes possible when the local strategy allows for a return to minimal routing before the obstacle is cleared. The message may then get routed back into the blocked region, a sequence that can cycle over and over.

However, to even approach this state of affairs is beyond the fault handling capability of oblivious or minimal adaptive routers. The problem is not due to a characteristic of non-minimal adaptive routing as such but arises with any algorithm dependent on local information in a sufficiently faulty network. Source routing, in which an arbitrary path is predetermined based on global knowledge of the network configuration, can route messages across an arbitrary network configuration that results from a series of faults. However, such an algorithm is static with respect to a given message and requires prior knowledge of the network. The resultant lack of adaptivity does not allow for dynamic fault handling.

The need for non-minimal routing in a faulty network is independent of the basic routing algorithm incorporated into the network. Therefore, in order to provide robust fault tolerance in interconnection networks that utilize other than non-minimal routing algorithms, some sort of additional exceptional mechanism that allows for non-minimal routing must be added. It follows that a non-minimal adaptive router will have an advantage as the basis for fault tolerance, requiring the minimum amount of incremental circuitry and effort to implement a fault tolerant design.

1.4 Fault Tolerant Chaotic Routing

Chaotic routing, a non-minimal adaptive packet routing scheme, is the subject of this design study. While the Chaotic routing algorithm provides for a robust basis for network redundancy management, the use of such an algorithm is not a sufficient condition for fault tolerance. It will be shown that the addition of a small amount of circuitry, interconnect and functionality can greatly enhance the fault tolerance of a Chaotic network. Fault detection is provided along with mechanisms that support fault diagnostic procedures. A method for network reconfiguration that marks faulty and out-of-service links is defined. The routing algorithm is modified to account for inoperative links and to compensate as smoothly as possible for the resulting nonuniformities in the network. The interaction that is required between router, network interface and system software in order to provide for effective fault management is defined.

Local information is used to detect errors caused by faults. However, the actual fault may be remote to the detecting node. A corrupted packet may not be detected until it has reached its destination, long after proximity to the actual fault has been lost. Fault identification and diagnosis requires a global approach if it is to be effective. The Express Broadcast Network

(EBN), a low overhead control network orthogonal to the data network is introduced. The EBN provides a mechanism for fine-grained global control of the interconnection network that has very low latency and low cost in network resources while providing highly reliable service. In the fault tolerant Chaotic routing network, the EBN is used for the synchronization and control of global fault management procedures.

The effectiveness of the enhancements which make up the fault tolerant Chaotic routing architecture are examined and it will be shown that reliability can be improved by an order of magnitude or more at a cost that is but a small fraction of the cost of the basic Chaotic routing network.

2 Chaotic Routing

Multicomputer networks usually consist of a set of nodes connected by a regular set of point-to-point communication channels which form a high bandwidth local network. An example of the structure of a typical network, in this case a two-dimensional mesh, is shown in Figure 1. Each node consists of a router and processing node. The processing node contains the processor, network interface and memory.

Chaotic routing is an implementation of a packet-switched communication network. A packet-switched network splits messages into blocks of fixed maximum size called packets. Smaller sizes can generally be accommodated. The packet becomes the unit of communication and has an existence within the network that is independent of the other packets that make up the message. The message is reassembled by the network interface at the destination processing node. Packet switching efficiently deals with a heterogeneous communication environment, in this case a computer network in which messages of widely varying lengths will be passed and in which some nodes will be very active while others will use the network much less intensely. Packet switching smooths out the network traffic flow by allowing for the sharing of network links. During the transmission of a long message, packet switching allows other message streams to compete for and alternate control of common network resources.

Chaotic routing uses a non-minimal adaptive routing algorithm [Konstantinidou 91]. Chaotic routing's identifying feature is the use of randomness to provide probabilistic protection from livelock. This simplifies the routing logic by removing the need to deterministically guarantee message delivery. Chaotic routing has been shown to be deadlock and livelock free on hypercube and many other types of networks including all k-ary d-cubes.

An examination of the Chaos router data path for a two-dimensional router, shown in Figure 2, demonstrates the function of Chaotic routing. The main router elements are the input and output frames, the crossbar and the multiqueue. Each input and output frame and multiqueue slot is a fifo buffer capable of holding one full fixed-size (20 flit) packet, 20 flits being the maximum packet length for this network. There is one input frame and one output frame for the positive and negative direction in each dimension plus an injection (input) frame and an ejection (output) frame for the router/processor interface for a total of five frames of each type. Simulation showed five multi queue slots to be optimal for 256 node two-dimensional mesh or torus networks. [Bolding 93].

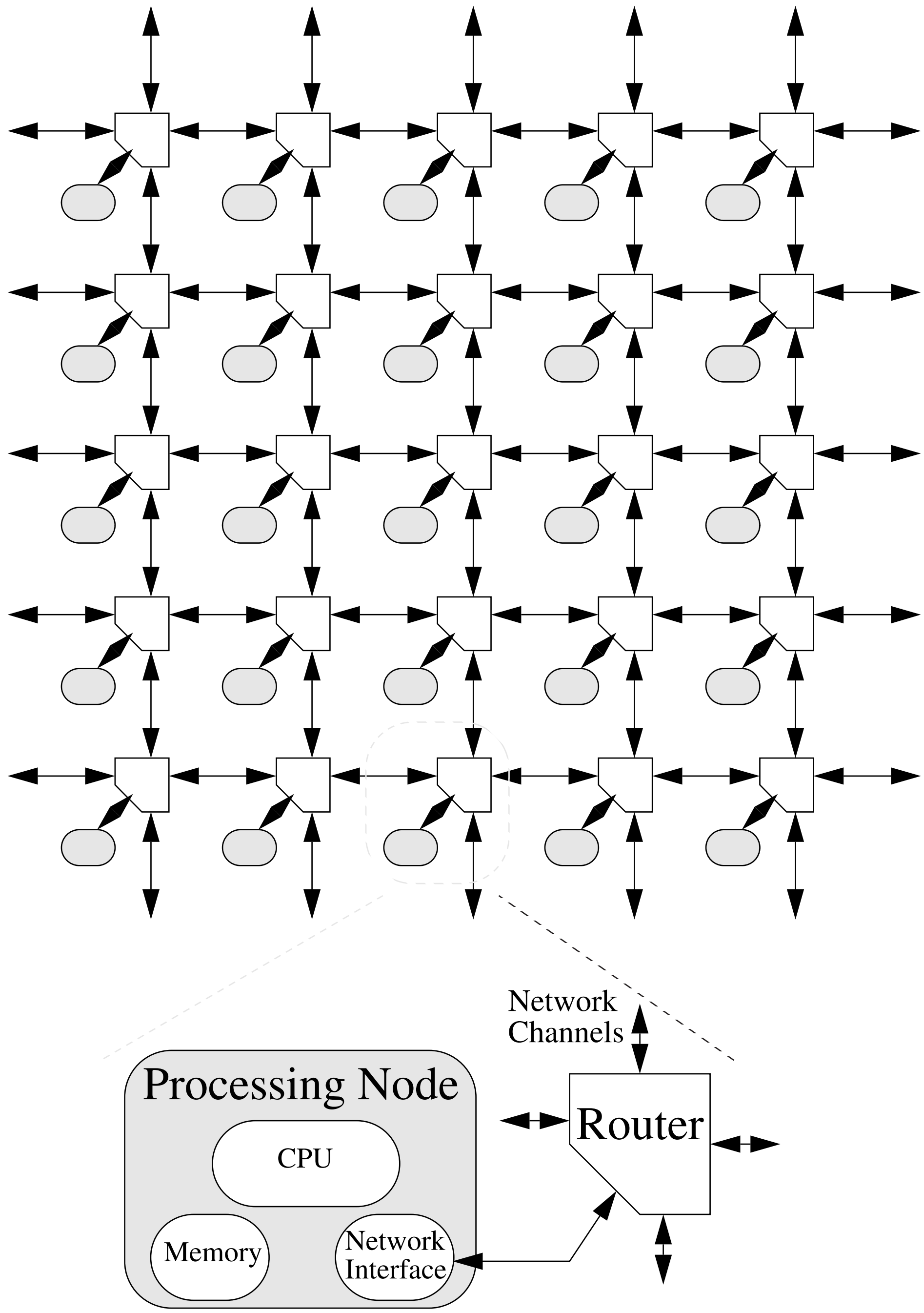


Figure 1: Mesh-connected interconnection network, with expansion of basic node.

During basic operation, a packet entering a router input frame is routed through the crossbar to an output frame in a direction that makes positive routing progress. This routing decision is generated by a minimal adaptive algorithm that allows a choice of any profitable direction for routing. The packet header is updated as the packet exits the router to reflect

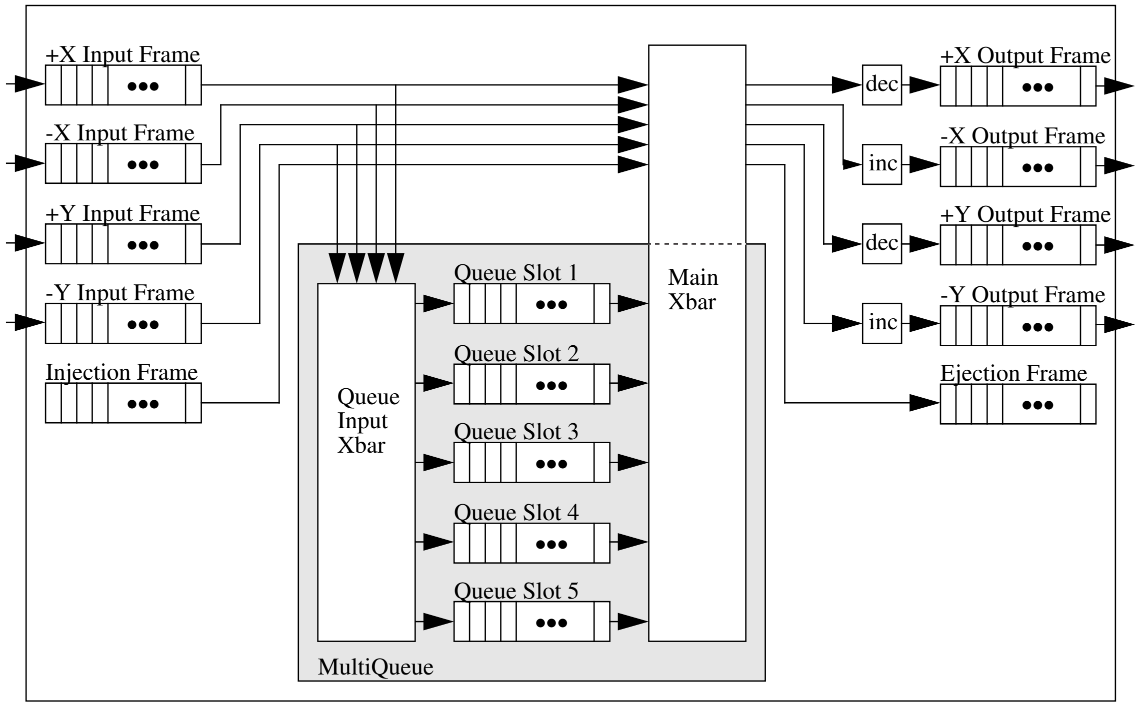


Figure 2: Two-dimensional Chaos router data path.

progress through the network. Virtual cut-through allows a packet to be routed as soon as its header is received and decoded but before the packet is fully received and buffered by the router. This may result in a packet being spread across several routers.

The multiqueue is a central buffer which stores packets that the router is not able to profitably route. The deadlock prevention protocol for the channels may also cause the entry of packets into the multiqueue. Packets in the multiqueue are given preference over packets in the input frames when a packet is to be selected for routing to a free output channel. When the routing load is moderate, the multiqueue buffering smooths the flow of packets through the router and allows the minimal routing algorithm to be effective even during transient periods of congestion. Whenever the multiqueue becomes full and no packet in the multiqueue can be profitably routed, the non-minimal routing algorithm activates. A packet is selected from the multiqueue *at random* to be routed out the next available output channel. This process of performing non-minimal routes is called “derouting”.

The ability to deroute packets as needed eliminates any node-to-node dependencies that can contribute to deadlock. The packet exchange protocol defined by [Konstantinidou 91] guarantees that given a finite number of bidirectional channels in each node, no channel will ever deadlock. Together, these two characteristics make chaotic routing deadlock-free.

Livelock freedom in the chaos router is probabilistically, but not deterministically, guaranteed. It has been shown for all finite-sized networks that the probability a packet remains undelivered after *t* seconds goes to zero as *t* increases [Bolding 93] [Konstantinidou 91]. The proof follows from the combination of a finite chance at each step of a packet being routed closer to its destination together with the finite nature of the network. As the number of routing steps increases, the chance of continued failure to reach the destination drops to zero. Bolding has also shown through simulation studies that for most networks the probability at any step of being profitably routed is generally greater than 0.90. This places a reasonable limit on delivery time for networks of practical extent.

2.1 Fault Tolerance

The combination of redundancy in the network together with the non-minimal routing algorithm provides for a natural degree of fault tolerance in the Chaos routing network. A persistently unresponsive link or router will not allow the output frames in the routers connected to the faulty component to empty. A router will simply treat such a frame as it would any unavailable frame and will route packets in other profitable directions, or lacking such a direction, will buffer the packets in the multiqueue. Ultimately, such a packet will either be routed in a profitable direction that later becomes available or it will be derouted. In any event, the unusable channel will be bypassed. At each step in the network, a new routing decision process for the packet will be made.

The nonuniformity in the network caused by a failed component can create an island of congestion that affects neighboring routers. As the network adapts to the congestion, packets will be derouted by routers not actually touching the failed component. The effect will be like water flowing around a rock in a river. The greater the flow, the farther upstream the perturbation to the even flow will extend. Packets that are within the congested area will be delayed as they work their way around the obstacle.

2.2 Shortcomings of the Basic Chaos Router

While faults can be bypassed either through alternate minimal routes or by derouting, a more efficient scheme for fault management is desired. Derouting, in particular, is only activated to extend the network performance when more efficient mechanisms lose their effectiveness due to the traffic demands on the network. To depend on the derouting mechanism to avoid faulty components places portions of the network permanently in its most inefficient operational regime. A means of incorporating knowledge of identified faulty components into the routing decision would help maintain good performance in the presence of faults.

Packets in an output frame connected to a faulty, unresponsive component will not be able to progress through the network because the only way for the packet to exit the frame is through the unresponsive component. This is necessary if the basic mechanism for avoiding faulty components is to work. The faulty direction must appear congested, i.e. the output frame must be full. A more robust scheme should provide an alternate method of mimicking congestion without requiring that a packet be sacrificed.

The paradigm of "fault as congestion" does not cover all fault manifestations. Packet data can be corrupted or a packet can be dropped without causing any network backup that would appear as congestion. Transient faults can disrupt reliable packet delivery in ways that don't affect the routing decision process. The number of possible fault effects that can be handled needs to be expanded through improved fault detection methods. Closely associated are fault diagnostic procedures that provide the inputs for a network reconfiguration procedure. In other words, a unified set of fault management procedures is needed for fault tolerance to be enhanced.

3 Motivation

Although extremely high reliability is desired in computer systems, no system can be built that will operate with 100% reliability. Failures may be hard (permanent) or soft (transient) and may occur for many different reasons including the following:

- Physical wear-out.
- Marginal specification and implementation that results in electrical problems.
- Latent manufacturing flaws not caught by screening and testing.
- The effects of the outside and system environment.
- Human error.

A balance must be struck between the need for reliability and the willingness and ability to devote the necessary resources to achieve that goal. The reality is that the interconnection network is a small part of the overall multicomputer system and a small investment in resources may be all that is available to devote to enhanced network reliability. Large gains in network reliability will translate into small improvements in system reliability. While this doesn't mean that the reliability of the routing network can be ignored, it does mean that the routing network will receive only a small piece of any marginal resources or design effort that can be dedicated to fault tolerance. Also working against fault tolerance in the router are the performance requirements. Maximum throughput and minimum latency are strong requirements, therefore any additions to the router must avoid increasing cycle time or pipeline depth. The result is that the design space of interest avoids wholesale changes to the network, carefully picking the spots in which the maximum return can be achieved with only incremental cost. Working in favor of this design space is the natural redundancy of the network which can be utilized to provide fault tolerance without the addition of redundant resources to the network.

3.1 Fault Model

The single sequential fault model that is used to analyze the hazard due to network faults assumes that faults will occur at a rate much slower than the response time of the network to a detected error. The single fault assumption precludes consideration of an arbitrary collection of faults that, at an extreme, could reroute the network into a different configuration or reorganize a set of transistors inside a chip into a new and different circuit. The single fault assumption does not eliminate consideration of complex errors caused by the fanout of effects from a single fault. Transient errors are included in this fault model definition along with permanent faults. An upset due to a transient error is usually the result of a bit flip or the improper reception of data by a storage element due to operating or environmental conditions.

A study of transient upsets in a microprocessor [Elder et al. 88] defines three categories of error response: bit errors, word errors and complex errors. A bit error is the result of a data register bit flip. A word error is the result of a bit flip in a register holding an address or an address pointer. A complex error is harder to characterize simply but is caused by an upset in the control logic and results in a change in the operating sequence or in the control outputs of the device. The complex error can include manifestations of the other error types. This characterization of errors turns out to be convenient in defining the errors due to routing network faults. In a routing network, a bit error results from an error that affects the packet data. A word error affects the routing control resulting in the misroute of a packet. A complex error affects the control or sequence of operation.

Within the interconnection network, faults may occur in either the links or the routers. Link faults are usually caused by connector problems as the wires in cables and circuit boards are generally highly reliable. A link fault will corrupt the signal on one or more lines. A corrupted data line results in a bit error. A corrupted control line can generate a complex error.

Within the router chip, a single hard fault may be represented as an open or shorted line or as a faulty transistor. An upset can be represented as a bit flip in a latch. Faults or upsets within the router data path will generally be manifested as bit errors. In the routing decision logic, data joins the control path and word errors can be generated as can bit errors and complex errors. In the router control logic, complex errors will be prevalent.

The single fault scenario includes macro problems that fault such global systems as the power supply or the clock. This category also includes gross physical problems from a single source. An example of such a failure would be an unmated connector or connector failure that disconnects an entire link(s). Another example would be a router die that is cracked due to physical stresses. This type of problem will generally have catastrophic effects on the link or router and generate errors of all three categories.

Network failures can be categorized in a more symptomatic manner without reference to fault location. Data in a packet can be corrupted resulting in bit errors. Channel control can fail to observe protocol or channel control can follow contextually incorrect protocol, i.e.

packet can be stretched or truncated resulting in complex errors. Channels can lose synchronization resulting in errors of all types. Packets can be misrouted resulting in word errors. Packets can be dropped or multiple copies of packets can be propagated resulting in complex errors. These symptoms may appear in combination. For example, a dead channel can manifest itself as the corruption of data, failure of channel control to observe protocol and/or dropped packets.

3.2 Fault Detection

Fault detection, the process of recognizing that a fault has occurred, is the first step in fault management. Strictly speaking, faults, which are in the physical domain, are not detected. Errors, which exist in the information domain, are detected when circuit conditions *activate* the fault and cause improper behavior. For the purpose of this study the terms "fault detection" and "error detection" can be used interchangeably. Fault detection can be closely related to the physical faults. A scheme with tightly coupled cross-checking redundant blocks would put the detection mechanism in close proximity to the physical fault. However, the approach used in the fault tolerant Chaotic routing architecture depends upon the detection of errors at a more abstract level. This approach turns out to be significantly less costly but is only marginally less effective at fault detection than duplication with cross-checking.

Fault detection mechanisms within the Chaotic routing network are described later in detail but can be enumerated quite simply:

Both a checksum and parity are generated for all packets in the network. Parity protects routing information in the packet header and is checked and updated in each router. The checksum protects the remaining static packet data and is checked in the destination node.

A channel protocol checker is used to provide a sanity check on transactions. This includes a timeout counter to detect unresponsiveness, a flit counter that detect packets of greater than maximum length and a context checker on the channel control signals that checks for inappropriate actions.

Packet reordering buffers located in the network interface are used to detect late, dropped or duplicate packets. Packet reordering is required because of the potential for out-of-order packet delivery that exists in a non-minimal packet router. Normal behavior assumes timely and coordinated delivery of packets in a multipacket message. Exceptional behavior, as manifested by missing or extra packets, indicates a possible system malfunction.

Single packets that encompass an entire message will not be processed by the reordering buffers and therefore, if missing, will not be detected via any interconnection network hardware. Any detection will be at the system level by means of a higher level mechanism. This can be through a protocol that requires an acknowledgement of message receipt or by means of a timeout or sanity check that depends upon an understanding of the software context such that the absence of the message indicates a problem.

Undeliverable packets within the network will eventually be detected when a system drain occurs as part of the fault management procedures. The system software uses the evidence provided by such packets to determine that some nodes may be unreachable.

3.3 Fault Coverage

Fault coverage is the average of the probabilities that failures within a fault class will be detected weighted by the probability of occurrence of each fault class. Fault coverage is an important factor in the estimation of the reliability of a fault tolerant system. The operative assumption is that an uncovered fault may lead to a partial or full system failure while a covered fault will be detected and the situation will be corrected, leaving the system functional. Other issues, such as the amount of redundancy available in the system, may enter into the reliability equation. However, for a large multicomputer with a highly redundant interconnection network in which large numbers of faults will not be allowed to accumulate, fault coverage becomes the key issue. Reliability is typically defined as $R(t) = e^{-\lambda t}$ with λ equal to the failure rate. The following equation relates coverage to the failure rate of the fault tolerant system and thus to the reliability:

$$\lambda_{FT} = (1 - \text{coverage}) \times \lambda_{baseline}$$

Thus, if coverage of 0.9 is achieved then the failure rate of the fault tolerant system will be 10% of the baseline failure rate. For coverage of 0.99, the failure rate will drop to 1% of baseline.

The determination of fault classes is somewhat arbitrary but depends upon locality, functionality and expected response to faults. The router chip encompasses three fault classes and the links represent a fourth. The probability that failures will occur within each fault class is implementation dependent and can only be estimated based on some assumptions as to fault behavior.

An examination of the Chaos router chip, considering transistor count, die area and the amount of wiring, shows it to be roughly 75% datapath, 10% routing decision logic and

15% control logic. A tabulation of the storage bits within the Chaos router chip shows that each router contains 15 fifos, each of which holds 20 flits of 16 bits for a total of 4800 bits. There are approximately 500 other storage bits within the router that are assumed to be distributed between the routing decision logic and the control logic in the 10:15 ratio suggested above. This associates about 90% of the storage bits with the datapath. A reasonable approximation for the effectiveness of fault tolerance in the router considers hard and soft faults within the datapath, routing decision logic and control logic as different fault classes. Estimates for fault coverage in these classes can then be applied and overall system coverage determined.

Faults within the Chaos router chip are assumed to occur randomly across the entire chip. This means that for each fault class, hard faults within that class will occur in proportion to the number of devices and amount of routing associated with that fault class. Soft faults will occur in proportion to the number of storage bits that can be associated with a fault class.

Within the network links, there is assumed to be some physical separation between data and control bits in order to minimize complex errors due to shorting faults between data and control lines. Faults will occur with a uniform distribution across the link signals. On the Chaos router chips, about 80% of the pins are associated with connections to network links. The connection points from chip to board and from package to die bonding pad are generally recognized as failure prone locations. Faults in the Chaos router chip from the die on out can be associated with the network link fault class based on the high associativity of chip connections with the link signals and the expectation that faults on either will be manifested as errors of a similar nature.

The distribution of the probability of occurrence of router faults versus link faults is difficult to estimate but turns out to be a moot point because of the similarity in coverage numbers that will be generated between the link class and the classes associated with the router circuitry. Basic connector reliability is high relative to that of a complex integrated circuit such as the router chip. However, in a large network, the connectors are probably the components most subject to mechanical damage, human error and stress as described in Section 3.4.3, which could skew the fault distribution in the direction of the link class. Table 1 summarizes the network fault classes.

Table 1: Chaos network fault classes.

Fault Class	% of router
Router - hard faults	
datapath	75
routing decision logic	10
router control logic	15
Links	
data lines	80
control lines	20
Router - soft faults	
datapath	90
routing decision logic	4
router control logic	6

3.3.1 Router Datapath Coverage

The datapath consists of input, output and multiqueue fifos along with the crossbar and associated routing that joins these blocks together. Faults in the datapath will generally show up as bit errors. Bit errors are detected with high probability by means of both checksum and parity. Coverage for bit errors anywhere in the system is near 100%. This holds for both hard and soft faults. It is possible but with low probability under the single sequential fault assumption that the checksum or parity detection can be defeated by a single fault that generates a common mode error within the datapath. However, multiple bits would have to be affected for the corrupted data word to be aliased into a word with no apparent errors.

3.3.2 Routing Decision Logic Coverage

Routing decision logic takes packet header data which contains routing information and generates control signals which control the path of the packet through the router. For packets which get buffered in the multiqueue, a data base is generated and maintained in the scoreboard where it is used to determine the future routing of the packet. The routing decision logic consists of the header decode logic, which determines the direction a packet will follow through the crossbar, and the multiqueue scoreboard. When a fault simply alters data, the result will be a word error, unless changing the data removes a pointer to the packet in which case the result will be a dropped packet, a complex error. In a similar manner, a fault can create a pointer to a phantom packet. Since routing decisions are time and context dependent, a word error will not necessarily appear as unusual behavior. Many word errors will remain unobtrusive though a few will generate detectable problems. For example, if a router always routes packets away from the local delivery channel or the sole remaining channel to another router, the lack of delivery will be eventually be detected. The diagnostic procedures have a good chance of detecting word errors. Routing decision logic faults can generate other types of errors. Bit errors can be generated as data words collide in the crossbar due to faults in the crossbar control logic. Complex errors and bit errors result from truncated or stretched packets. The effect of faults in random logic depends completely upon the low level implementation of the circuitry and is difficult to predict but an assumption that the effects of some faults in the routing decision logic could fanout into the router control and result in large scale control problems is prudent. This type of error is likely to result in various types of complex errors, in particular channel control problems.

Any coverage estimate for word errors must be qualified. Most word errors will be harmless and continued operation of a router that is nominally faulty but still active may actually have less negative impact on the network than would removal of the router from the system via reconfiguration. The ability of the network to function in the presence of router word errors could be considered an inherent form of fault tolerance. A soft fault that results in a word error should have no lasting impact because the packet is unaltered, routing information in the packet header is consistent with the packet location, and the packet can proceed from its new location without encountering the same problem again. The mechanisms for detection by the reordering buffers and for the detection of undeliverable packets within the network by means of a system drain should catch most word errors that actually interfere with successful packet delivery but detection latency may become an issue.

Complex error coverage will depend upon the manner in which the problem is manifested.

Truncated or stretched packets should be detected via bit error detection mechanisms. Multiple or dropped packets will not be detected by the routing network because the routers have no context from which to base a judgement concerning the correctness of the existence or lack of existence of a packet. That task is reserved for the network interface or the operating system. It is expected that most faults affecting channel control will result in clearly improper behavior. Byzantine behavior due to faults in which an error is manifested as behavior that is improper for the specific situation but not outside the realm of legal activity as determined by the channel protocol checker, is possible but is expected to be rare.

3.3.3 Router Control Logic Coverage

The router control logic has overall control of the sequencing of the router operations. This involves controlling the ordering and sequence of dimensions that are visited along with many housekeeping tasks such as fifo control and channel control. Faults in the router control logic will be manifested as all three of the error types. In particular, control of the fifos can result in bit errors while channel control can result in complex errors.

3.3.4 Link Coverage

Link failures will probably be detected as bit errors in proportion to the number of data bits in the link. Chaos uses 16 data bits and 4 control bits per link, therefore the 80% of single line or pin faults that result in bit errors will be easily detected and covered. Errors involving open or shorted control bits may result in complex errors with the attendant lower coverage. Catastrophic failures involving the loss of the entire link because of connector failure will be easily detected.

3.3.5 Network Coverage

The major problem encountered in this analysis of coverage concerns the relationship between the error detection mechanisms in the fault tolerant Chaos routing network and the bit, word and complex error classifications considered in the discussion of the various fault classes. Fault classes are physical classifications of the network while the error detection occurs in the information domain and the mapping between the two is complex. Fault injection and simulation can provide an explicit mapping but that cannot be done using a high-level conceptual design. Error detection and its effectiveness is discussed in Section 5 and a conservative mapping of error detection into fault coverage is described.

3.4 Reliability Estimation For a Chaos Routing Network

The following sections provide an analysis of the base failure rate for the Chaos routing network. This information can be combined with the fault coverage estimates for the fault tolerant network to provide a reliability comparison of the base routing network versus the fault tolerant network in absolute terms.

3.4.1 Failure Rate Estimation for Hard Faults

The following calculations use the reliability estimation equations and data defined by Mil-Hdbk-217F [MH217 91]. This handbook is generated through the compilation and analysis of large amounts of reliability data on all types of electronic components. Formulae to calculate expected failure rates for various part types are defined and the various factors incorporated into the formula such as environment, part complexity, package type, etc. are tabulated. Reliability is defined as $R(t) = e^{-\lambda t}$ which assumes an exponential distribution of fault arrival times with constant failure rate λ. This distribution also defines the Mean Time To Failure (MTTF) as $1/\lambda$. The reliability of a system is defined as $R_s = e^{-[\sum \lambda_i]t} = e^{-\lambda_s t}$, where λ_s is the summation of the individual failure rates of all system components. The estimates given in the handbook always trail current technology because of the retrospective nature of the data. However, they give a good relative baseline and order of magnitude estimate. For this example a 1024 node mesh network is assumed. Only the reliability of the network components is considered in this analysis, not the reliability of the full parallel computer.

The Chaos chip is a custom designed 1.2 micron feature size CMOS device. It is contained in a 128 pin PGA and the die itself is approximately 1.0 cm^2. It contains about 75,000 transistors which translates into about 20,000 gate-equivalents. This falls between two 217F categories, “Gate/Logic Arrays and Microprocessors” and “VHSIC like microcircuits and VLSI CMOS (60,000 gate +)”. The calculation using the “Gate/Logic Array and Microprocessors” formula gives a higher failure rate so it is used as a conservative approach. The expected failure rate calculations are shown below.

For Gate/Logic Arrays and Microprocessors, the formula is as follows:

$\lambda = (C_1\pi_T + C_2\pi_E)\pi_Q\pi_L$ failures/10^6 hours where:

C_1 is the die complexity failure rate.

C_2 is the package failure rate.

π_T is the temperature factor (junction operating temperature).

π_E is the environment factor.

π_Q is the quality factor.

π_L is the learning factor.

For the Chaos chip the values for these parameters are shown in Table 2. When appropriate, a range of values is given. Many of the parameters are insensitive to changes in the chip. The significant variables turn out to be the operating temperature, the quality screening level and the maturity of the design. Ignoring the screening level for which improvement carries a very high price, the high and low failure rates for the chip are within a factor of three of the baseline rate.

Table 2: Chaos chip reliability parameters.

Parameter	Low	Baseline	High	Discussion
C_1	0.16	0.16	0.16	10,001 ≤ # gates < 30,000 With the Chaos router having 20,000 gates, this parameter is insensitive.
C_2	0.053	0.053	0.053	The baseline value is for a 128 pin chip. The next step comes at 180 pins.
π_T	0.16	0.35	0.71	T = 35°, 55°, 75° C.
π_E	0.5	0.5	0.5	Baseline is “Ground, Benign”, a lab or business environment
π_Q	2	10	10	Heavy screening vs. commercial quality
π_L	1	2	2	Part in production 2, 0.1, 0.1 years
λ (failures/ 10^6 hours)	0.104	1.65	2.80	

The reliability of the circuit board components also affect the system reliability. Consider a system built using backplanes containing 16 routers. Each board will have 100 pin PCB connectors for each of 16 processing node boards. There will be 16 additional 100 pin PCB connectors for external connections. A PC board has 5000 plated through holes. A 1024 node system requires 64 such boards.

For a 100 pin PCB connector, the formula is as follows:

$\lambda = \lambda_b \lambda_K \lambda_P \lambda_E$ failures/10^6 hours where:

λ_b is the base failure rate.

λ_K is the mating/demating factor.

λ_P is the active pin factor.

λ_E is the environment factor.

Values are as follows:

$\lambda_b = 0.00028$ (connector temperature = 30° C.)

$\lambda_K = 2.0$ (0.5 to 5 cycles/1000 hrs.)

$\lambda_P = 25$ (100 pins)

$\lambda_E = 2.0$ (Ground, Benign environment, not Mil Spec quality)

$\lambda = 0.028$ failures/10^6 hours

For a PC board with 5000 plated through holes, the formula is as follows:

$\lambda = \lambda_b(N_1\pi_C + N_2(\pi_C + 13))\pi_Q\pi_E$ failures/10^6 hours where:

λ_b is the base failure rate.

N_1 is the quantity of wave soldered PTHs.

N_2 is the quantity of hand soldered PTHs.

λ_C is the complexity factor.

λ_Q is the quality factor.

λ_E is the environment factor.

Values are as follows:

$\lambda_b = 0.000041$ (Printed wire board)

$N_1 = 5000$

$N_2 = 0$

$\lambda_C = 1.8$ (5 layer board)

$\lambda_Q = 2.0$ (not Mil Spec quality)

$\lambda_E = 1.0$ (Ground, Benign environment)

$\lambda = 0.738$ failures/10^6 hours

Therefore, for each board, the failure rate is as follows (ignoring sensitivities except for the Chaos chip) :

$\lambda_{Board} = 16 \text{ x } \lambda_{Chaos} + 32 \text{ x } \lambda_{Connector} + \lambda_{Interconnect}$

$= 26.4 + 0.896 + 0.738 = 28.03$ failures/10^6 hours (λ_{Chaos} baseline).

$= 44.8 + 0.896 + 0.738 = 46.43$ failures/10^6 hours (λ_{Chaos} high).

For the full network:

$\lambda_{Network} = 64 \times \lambda_{Board}$

$= 1.79$ failures/10^3 hours (λ_{Chaos} baseline).

$= 2.97$ failures/10^3 hours (λ_{Chaos} high).

By far, the greatest component of the failure rate is associated with the router chip. The overall MTTF is 557 hours for the baseline network and 337 hours for the network using a less reliable Chaos chip.

3.4.2 Failure Rate Estimation for Soft Faults

The occurrence of transient errors in a system has been estimated at anywhere from 6 times the hard error rate [Iyer & Hsueh 90] to 30 or more times the hard error rate [Siewiorek et al. 78]. While many causes of transients are known, transients, by definition, are ephemeral and troubleshooting can be difficult. Among the causes are electromagnetic interference (EMI) from the system or outside environment, power supply instability, marginal parametric characteristics at a device level that results in data ambiguity, failure to meet device setup and hold requirements, single event upsets (SEU) caused by the decay of radioactive materials in device packaging or from environmental (cosmic ray) radiation and subtle design flaws or protocol errors that create data conflicts.

For a rough order-of-magnitude analysis, the Chaos routing network is treated as a large distributed SRAM and the reliability with respect to transient upsets is calculated. The comparison is very rough because of technology differences. SRAMs have a specific technology dependent sensitivity because of their utilization of sense amps and signal swings that are a fraction of full rail values. In contrast, the Chaos router latches utilize non-array type full rail logic. However, other comparisons work in favor of the SRAM in this comparison. The Chaos router uses dynamic latches rather than static. The SRAM is monolithic and the interconnect is very carefully designed and controlled whereas the Chaos network is distributed and subject to outside influences. Problems such as clock skew are of utmost importance for Chaos.

Each Chaos router contains 4800 bits of storage in its fifos. In a 1024 node network the total number of bits is 4,915,200. State bits and other latched values are disregarded. A representative value of 10^{-8} errors/bit-hour for the bit error rate is assumed. The calculation is performed for a full network and for a network with buffers 1/3 full. Unless the network is fully saturated with packets, a good percentage of data buffer upsets will have no effect because they will occur on empty buffers. The results are consistent with the expected rate of transients relative to the hard failure rate. The estimation gives $\lambda = 16.4$ failures/10^3 hours for the 1024 node network that is 1/3 full yielding an MTTF of about 61 hours, 11% of the MTTF of 557 hours for hard failures. For the network that is full, $\lambda = 49.2$ failures/ 10^3 hours yielding an MTTF of about 20 hours, 3.6% of the MTTF for hard failures.

3.4.3 Potential Network Problems

The previous sections have abstracted the potential failures into estimates of failure rates. Here some of the potential design problems specific to large networks that can contribute to unreliability are described.

A large network is by nature a physically distributed system. The sheer number of chips, boards, links, connectors and other components becomes enormous. The example that was analyzed had 1024 nodes organized into 64 racks of 16 nodes each. Tying the system together is a difficult task. Maintaining a stable, uniform environment is a key issue. Timing issues and clock skew are accentuated by the distributed nature of the system. The normal parametric distribution across a large number of parts can result in some outlying parts whose marginal performance affects the system. This effect is multiplied by power supply variations and power dissipation hot spots that can create “fast” or “slow” regions of the system. Clock distribution and synchronization, which require careful design at the chip or board level, are even more important and difficult for a large system. The Chaos router was built with a cycle time of 15ns. The goal for the next version is 6ns. Other router designs using more aggressive technology have a 2ns or 3ns cycle time. The dual trends toward the growth in system size and the increase in clock rate combine to shrink the margin of error in system timing.

Human error is difficult to quantify as a reliability factor but must be considered as a source of unreliability. If a small computer malfunctions, one of the first things to check is the power and all of the connections. In a larger network, the number of connectors to mate and boards to seat is much larger and a bad connection may affect only a fraction of the system. In fielded systems, when failures are examined and boards and black boxes pulled because of some indication of failure, it is common for the result of the examination to be “No Failure Found.” It is suspected that bad connections, whether caused by humans or by environmental conditions, are the source of many of these problems and the simple act of manipulating the connectors and reseating the boards as part of the troubleshooting process alleviates the failure. Failures due to such factors as electrostatic discharge (ESD) can also be attributed to improper handling and human error.

Dutton [Dutton 89] in a discussion of connector reliability, states that 15% of automobile breakdowns can be attributed to connector problems. While the automotive environment is far more harsh than a scientific computing environment, some of his observations have general relevance. Connector unreliability occurs as pins tend to back out of the socket and contacts tend to spread, causing reduced contact which can result in intermittent problems that are particularly difficult to trace. In an automotive environment, these problems are the result of repeated mating cycles which are largely due to routine maintenance or diagnostic checks. In other words, the basic unreliability of a system may tend to exacerbate and create additional reliability problems. The addition of fault tolerance and diagnostic capability to a network may actually lower the base component failure rate by avoiding some of the physical manipulation that can degrade the system.

3.4.4 Summary

The failure rate for the fault tolerant network is not the component failure rate but the rate at which failures occur that are not masked by the fault tolerance, resulting in full or partial system failure. The baseline MTTF of the network for hard faults is estimated at 557 hours or about 23 days. The MTTF for soft faults is much less. The addition of fault tolerance to the interconnection network could improve the reliability to the point that regularly scheduled maintenance could be used to repair network faults without any unscheduled loss of system operating service.

4 Communication and Network Synchronization

In a multicomputer system, the most efficient algorithms are those that can be distributed across many nodes in order to take advantage of as much processing power as possible. The latency of the interconnection network adds significant overhead to a computation, reducing the efficiency of computations with many inter-node dependencies. So it is with fault tolerance. Diagnostic procedures that perform fault identification are quite simple and efficient when concerned only with the health of the immediate neighborhood of a router within the network. As with the Chaos routing algorithm, which uses only local information in making routing decisions, fault detection circuitry that works strictly with local information provides many advantages. Locality avoids a large amount of communication overhead and simplifies fault management procedures. Fault tolerance is more robust without a master fault tolerance controller that, itself, would have to be fault tolerant in order to avoid single point system failure scenarios. This type of design scales well as the multicomputer system grows. It also enhances fault containment should a particular fault result in an error that escapes from the node. An error that propagates away from the faulty node will quickly reach a node that is physically unaffected by the fault. Because the local nature of fault detection requires no byzantine agreement between nodes, that node will also be logically unaffected by the fault. Therefore, the fault detection capability in that node will be working at full speed, providing maximum probability of fault detection.

While localized fault management has much to offer, the network has a global nature that cannot be ignored. The paths that packets travel through the network may pass through routers remote from both source and destination. The traffic load that is applied to a node depends on the node's local environment *and* on global traffic patterns. The global nature also has an impact on fault detection and management. Diagnostic procedures generally require that the system under test be in a known initial state. However, any attempt to locally set the network state will be immediately perturbed by traffic from outside of the local region. It can also be the case that local diagnostics may only be of limited use. A missing packet may be anywhere in the network while a packet may not be detected as corrupted until it has moved far from the source of the problem. However, this doesn't invalidate the concept of local control of fault management. Through network-wide synchronization of the local procedures, the net effect can be of a global, but distributed procedure. This requires an efficient broadcast mechanism through which network synchronization and control can be maintained.

4.1 Broadcast Networks

Broadcasting in a distributed network is a complicated problem. Typically networks that are optimized for data transmission have very high bandwidth and relatively low latency in order to deliver messages efficiently between processing nodes. The network design and associated routing algorithms tend to be optimized to provide the best service for the passing of data between processors. Unfortunately, data networks do not necessarily provide good service for control messages, which have different network needs.

Data messages need networks that provide high-bandwidth, reasonably low latency, and reasonably reliable delivery for point-to-point communication. Routing algorithms designed for moving data usually send a single copy of a message through a single (possibly dynamic) network path. Failures in the network may cause loss of communication between nodes and loss of data until the faults are detected and the operating system reconfigures the network.

On the other hand, control messages require little bandwidth, but need very low latency and extremely reliable delivery for broadcast (one-to-all or all-to-all) communication. Control messages may be used to signal network barriers, changes in network configuration, the onset or termination of computation, or warning of network errors. They should be able to reach all connected nodes in a network in a timely manner regardless of faulty links. When control messages are sent across a network optimized for data transmission, they are treated in the same manner as all other messages. They are therefore likely to experience large delays in times of high congestion and they are subject to being lost by the network due to component failure. Designing a network to give priority to control messages over data messages helps alleviate the problem of congestion, but does not help when components fail. Also, most data networks do not directly support broadcast, so the broadcast of system control messages must be accomplished by a multitude of single-destination messages. There is also the fundamental philosophical and practical problem of using a network to control the test of suspect components which make up that same network. This situation exemplifies the Byzantine Generals Problem [Lamport et al. 82] in which malfunctioning components may pass conflicting information to various parts of the system.

In order to provide efficient transmission of control messages, a separate network, either physical or virtual, is needed. A small number of currently produced parallel machines have multiple networks for different functions. The Thinking Machines CM-5 computer includes three distinct physical networks: a data network, a control network, and a diagnostic network [Leiserson et al. 92]. The Fujitsu AP1000 also has three networks, one for data, one for control broadcasting, and one for synchronization [Ishihata et al. 91]. However, the additional hardware needed to build three separate high-bandwidth networks is very costly. The interconnection network, in its quest for high bandwidth and low latency, is usually designed to fully utilize available network resources (wires and pins). Therefore, a practical but alternate broadcast network must minimize such usage in order to avoid having a negative impact on network routing performance. IBM's Vulcan architecture uses a virtual broadcast network built on top of the data network's links to implement its fault-tolerance mechanisms [Stunkel et al. 94], avoiding completely the use of additional scarce resources. Unfortunately, adding a virtual network on top of the existing network complicates the critical path of the network routers. Thus, a low-cost, but physically separate control network is needed for a large-scale machine. The need for such a network for use in the control of fault tolerant Chaotic routing led to the design and development of the Express Broadcast Network, an inexpensive low-bandwidth, low-latency network which is extremely redundant and optimized for the broadcast of system-wide control and synchronization signals.

4.2 The Express Broadcast Network

The Express Broadcast Network (EBN) is designed to meet the critical requirements of a network to transmit system control messages, while costing very little to implement. The supported features are as follows:

- Low latency.
- Broadcast ability.
- Very high reliability.
- Performance independent of the data network load.
- Scalability.
- Reliability and performance nearly independent of failures in the network.
- Low implementation cost.

In order to achieve these features at a low cost, the network is restricted as follows:

- Very low bandwidth.
- Synchronous operation (It is not necessary that a global clock exists, but each node must be able to consistently count the passage of time.)
- Broadcast-only operation -- no single source/single destination operation.

The primary function of the EBN is as a broadcast network. A typical implementation of a broadcast network utilizes a common bus across the entire network or subset thereof. While this physical configuration is simple and best case performance is near optimal, it requires a complicated protocol to resolve contention and performance can drop sharply as the load grows. In addition, the common bus leaves the system vulnerable to faults. For these reasons, the EBN is implemented as a set of point-to-point links. This creates natural fault tolerance and reliability through redundancy and greatly simplifies channel protocol while maintaining good performance under all reasonable situations and acceptable worst case performance.

A simple method of implementation is to build a static tree structure into the network, where each node has in-degree one and out-degree of greater than one. A spanning tree could then be defined and each node would act simply as a repeater, copying its input message onto each of its outputs. Two limitations to this design limit its usefulness: each link is a single point of failure, and the tree has only one statically-determined source node. A failure on a link causes loss of communication to all of its descendants. Furthermore, the only node that can broadcast in this tree is the single source node. Cray Research's T3D uses a special network to implement barrier synchronization and broadcast primitives using static trees [Cray Research 93]. This allows multiple source nodes, but still suffers from single-point failures. To avoid these problems, the EBN incorporates a dynamic redundant tree structure.

Each node in the network maintains a "listening" mode, checking the incoming links for messages. Any node may become the root of the broadcast tree as the need arises. When a node desires to broadcast a message to the network, it simply sends a message to each of its neighbors and internally notes that it has generated a broadcast message. Each of the receiving nodes then marks itself as having received a message and copies the message to each of its remaining neighbors. This process dynamically constructs a new broadcast tree that touches each connected node. A node may receive copies of a message from more than one direction at once, which is explicit evidence of redundant minimum length paths in the tree. A node which has already received a message ignores any subsequent messages. All nodes in the network eventually receive the broadcast message. Moreover, the broadcast mechanism will automatically take advantage of the redundancy in the network, expanding around any number of faulty links to eventually reach all connected nodes in the network. This is illustrated in Figure 3, in which the broadcast reaches the entire network despite the presence of faults. The path redundancy is not limited to fault-free minimum length paths.

In a faulty network with all minimum length paths blocked between two nodes, the propagation scheme automatically reorganizes the tree and extends its depth as needed using alternate paths.

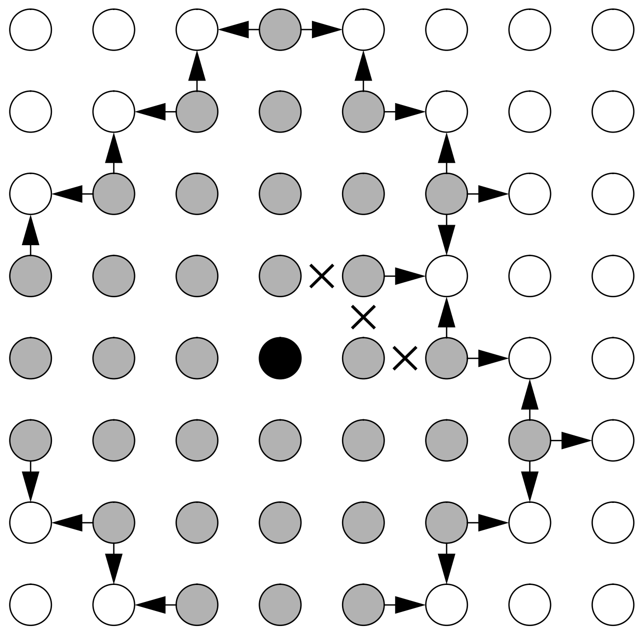


Figure 3: An EBN broadcast. A message originates from the black node which propagates the message to all of its neighbors. This illustration shows a broadcast that has spread four hops. As the broadcast message wavefront spreads, it bypasses faulty links (marked with X's) to reach the entire network.

The EBN has, in its simplest form, two basic communication primitives: *eurekas* and *barriers*. Eurekas [Cray Research 93] are one-to-all broadcast messages and are used when the entire network is waiting for a single event. Barriers are used when the network is waiting for all nodes to experience a common event. Barriers can be implemented by having all nodes broadcast eureka messages if they have *not* experienced the event. When no messages are broadcast, the barrier is complete. Synchronization details for these are presented in Section 4.3.3.

4.3 Implementation

The EBN, consistent with the low amount of bandwidth needed for control messages and the requirement that the implementation costs be kept low, can be implemented using only a single wire per link. While space (additional wires) and time (time encoded messages) can be balanced to provide greater bandwidth and a greater variety of messages, the single wire implementation can provide full EBN functionality. An implementation utilizing a pair of unidirectional wires on each link which allows only single bit messages is described as are extensions to multiple wire, multiple bit, and bidirectional wire implementations.

4.3.1 Basic Network Implementation

The EBN is constructed using broadcast nodes to join links. These nodes may be connected together in *any* shape that maintains connectivity of the network, regardless of the shape of the independent data network. The EBN routing algorithm is unaffected by the network shape. The connectivity of the EBN may be increased over the data network in order to minimize broadcast latency, or reduced in order to cut costs. However, for the sake of ease in physical layout, the network configuration of the EBN is envisioned to be the same as that of the data network so that the EBN wires can be bundled together with the data network wires. Moreover, because the EBN requires only a small amount of logic, the simplest approach is to design the EBN into the data network routing chips that are requisite in a parallel machine.

4.3.2 Network

In the most basic implementation, the nodes are connected with unidirectional wires (single source/single destination). To establish bidirectional communications between two nodes, two separate wires must be used. A set of unidirectional wires connecting each node and its associated processor is also necessary. Depending upon the desired functionality, a small number of control lines on the processor link may also be needed.

The EBN function is very simple. At reset, each router enters an inactive state. Each router ‘‘listens’’ to each incoming link. The router switches to the *active* state whenever any EBN input, including from the processor, becomes active. In the active state, the router rebroadcasts the incoming message to all of its neighbors by driving all outgoing links, including the processor link, to the active value. Every processor that is connected by the network is notified of the active state within a predictable amount of time that is independent of the data network load. The active state is actually a sequence of states. There may be one or two different transitions out of a given router state. When a choice of transitions is needed, a fixed window in time is defined, during which the EBN can signal through assertion or lack of assertion which transition will occur. Alternately, a single transition may occur at an indeterminate time, signaled by assertion of the EBN. Or a single transition may be timed without EBN input. The sequence of states will eventually lead back to the inactive state.

Since the work required at each EBN node is limited, a message in the EBN should propagate through each node with minimal delay. A pipelined design might implement this as two cycles: one for transmission across the wire and one for the broadcast. The maximum broadcast delay, the time for the assertion of the EBN by any node to propagate across the network, is bounded by the product of the node delay and the diameter of the network.

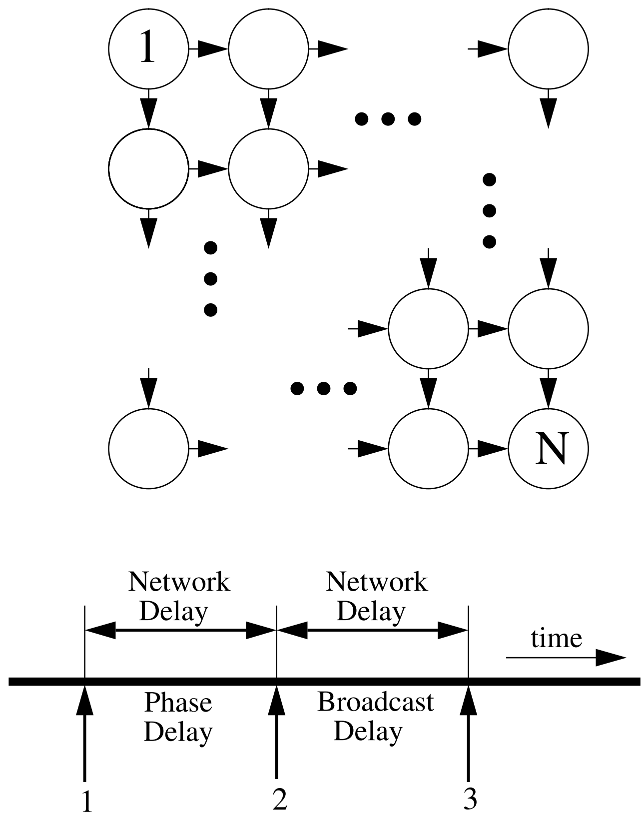
The broadcast delay across the network allows for the possibility that there may be multiple broadcasts from separate sources active across the network at a given time. Fortunately, with single bit messages, this does not present a problem because all messages received in a given state are interpreted the same way, regardless of their source node. It is only necessary that a method for the maintainance of network synchronization be followed so that messages will be interpreted by all nodes in the same manner.

4.3.3 Synchronization

Because the single wire EBN can only broadcast a single bit message, every node in the network must remain in at least loose synchronization in order to agree on the interpretation of a message on the EBN. The EBN itself is used to keep the machine synchronized. Each EBN node keeps track of the state of the network according to a predetermined set of state transitions, synchronized by the EBN. Because the propagation of the broadcast message is sequential and not instantaneous, synchronization is maintained by allowing time for the network to settle between broadcast messages. For example, the network starts with each node in the inactive state. When a broadcast is initiated on the EBN, each node switches to an active state as it receives the message. Each node that receives the broadcast will be tightly synchronized with the originating node but with a phase delay proportional to the distance from the originating node. After entering the active state, each node executes a fixed delay before any new transition is initiated. The required length of this delay depends upon the situation.

Define a *Network Delay* as the worst-case delay across the network. Consider the case in which EBN assertion (or lack of assertion) at a fixed time selects one of two possible state transitions. When the network is waiting for an assertion from an unknown source, it must wait at least two Network Delays in order to account for the phase delay across the network plus the broadcast delay of the EBN. This is illustrated in Figure 4. Node N changes state based upon a broadcast that it receives one Network Delay after the broadcast was initiated.

A node in the new state initiates a eureka broadcast if some condition is met at the time of the state transition. Node N responds to the change of state with a eureka broadcast which takes one additional Network Delay before it is received by Node 1.



1) Router 1 enters state A.
2) Router N enters state A, asserts EBN.
3) Router 1 receives EBN broadcast, moves to state B.

Figure 4: Synchronization delay. Each node must wait a period equal to the phase delay plus the broadcast delay.

However, if the transition to the current state was timed or was signalled by an EBN assertion and the next state transition is to be driven by another EBN assertion at an indeterminate time, then the delay in the current state can be short. Since the delay to the next state transition is not fixed and the transition itself is predetermined, there is no need to synchronize the transition to the previous transition as was the case with the previous example. The EBN can be asserted and the state transition can take place almost immediately; as soon as the condition that drives the transition is met. The other nodes will make the transition with whatever phase delay exists between them and the initiating node. There need only be enough delay to avoid conflict between the successive assertions of the EBN. During this delay, the node accepts no EBN inputs and drives no EBN outputs. This is because the EBN may still be asserted by neighbor nodes with trailing phase. Temporarily disabling the EBN in this manner ensures that EBN broadcasts from neighboring nodes that have not yet made the state transition will not cause unwanted state transitions in nodes with leading phase.

The proper phasing of events and state transitions is of particular importance in enhanced versions of the EBN in which multiple messages must be sorted by priority and filtered before a transition may be made. During the delay period, any action indicated by the message is performed. After the delay expires, the node, depending upon the protocol and the assertion or lack of assertion of a new synchronizing signal on the EBN, either returns to the inactive state and executes another delay in order to ensure that all nodes are inactive or makes a transition to another active state.

While the router will control the EBN along with the data network routing process, the node intelligence remains within the processing node. The function of the EBN is limited to providing communication services and maintaining network synchronization. EBN message content is generated by the processing node. While the router and processing node are connected by a standard EBN link, there is private communication associated with the EBN that is required between the two because of the fault detection functions within the router. Information generated by the router is used by the processing node in assessing whether or not to assert the EBN. This can require the use of additional dedicated control lines between the router and the processing node.

4.3.4 Multiple Wire Channels

The use of single wire channels limits messages to single bit messages. The content of these messages depend upon the state of the network at the time that the message is sent. More complex control schemes will require a greater choice of messages that may be sent when in a given network state. The use of multiple wire channels makes this possible. The number of distinct messages that can be sent from a given state is the number of distinct values that can be sent on the wires, w for log w wires.

However, the extension to multiple wires introduces a new problem. A simplifying assumption used in the design of the single wire EBN is that the meaning of a message is independent of its source: all messages mean the same thing. That is no longer the case, since two sources could simultaneously be broadcasting different messages in the multiple wire scheme. Prioritizing the messages solves this problem. A strict ordering of all possible messages must be made, so that a message is given strict precedence over all messages of lower priority. Whenever a node receives and forwards a message, it will continue to listen for messages of higher priority arriving on any of its incoming links. If a message arrives at a node that has already received a message of lower priority, the new message will be broadcast by the node just as if the first message had never been received. All nodes, including the sources of lower-priority messages that are preempted will ultimately receive the highest priority message. These sources can save their original messages for rebroadcast at a later time.

One problem remains: higher-priority messages may preempt others at any time, potentially extending by a significant amount the overall broadcast delay before the network settles. This can be fixed by constraining the generation of new messages. Not until a state transition occurs is the initiation of a new message possible after a node has received a broadcast message of any priority from another node. This reduces the worst-case time for the completion of a broadcast to proportional to twice the diameter of the network since the node farthest away from a broadcasting node may source a higher priority broadcast just before the original broadcast reaches it. Each of these broadcasts takes time proportional to the diameter of the network to complete, so the total operation is proportional to twice the diameter. Thus, each EBN node will have to maintain itself in “listening” mode for a long enough period to ensure that the current message will not be preempted before the node can make the transition to a new state.

4.3.5 Bit-serial Messages

Messages may be extended by time encoding the bits on each signal wire, an orthogonal process to extending the number of wires. Effectively, it is simply a tradeoff of time versus space in order to achieve a greater space of possible messages on the EBN. Time encoding allows an extended serial broadcast in the one-to-all broadcast mode, while the comparable extension in space, greatly extending the number of wires in the link, is clearly not feasible. A hybrid solution, with multiple wires and multiple bit messages, is also a legitimate extension of the EBN. One-to-all broadcast messages have a two phase structure. In the first phase, a node wishing to broadcast a one-to-all message broadcasts a setup message. Any collision of messages will be resolved through means of priority bits encoded into the setup message. At the end of phase one, a single node has established broadcast rights to the entire network, and all nodes are listening for that message. Each node will have marked as the parent direction the channel from which the broadcast request that ultimately prevailed was received. This sets up a pseudo-static tree for the ensuing one-to-all data broadcast. Any node that received more than one instance of the same broadcast signal on the same cycle may resolve the conflict arbitrarily in setting a parent direction. The second phase consists of the broadcast of the data portion of the message to the entire network, which will take place without conflict. When the source node relinquishes control at the end of the one-to-all broadcast, the network returns to the inactive state.

The lack of control lines associated with the EBN makes it necessary that a multiple bit message be either of fixed length or that there be a termination sequence to flag the end-of-message. Synchronization becomes complicated because of the possibility that messages of different priority will arrive at a router out of phase and on different channels. Any router may update the message that it propagates and set a new parent direction if it receives a new, higher priority message, therefore it is possible for a router to be the recipient of an overlapping stream of new messages from which it must decide on one to propagate. It will resolve this situation by propagating any fully received message of higher priority than any message either previously propagated or currently waiting to be propagated. Messages received while the router is in the process of propagating another lower priority message will be buffered for the next open propagation slot. The buffer will be dynamic with only the highest priority message preserved while all others are dropped. This protocol can result in a lot of frantic activity if multiple messages are active in the network. But with new message injection inhibited in any router that has received or injected a message into the network and with the damping of lower priority messages, the highest priority message will make monotonic progress across the network with only minimal incremental delay due to contention.

4.3.6 Bidirectional Channels

By using bidirectional wires, which can be driven by either connected node, the number of wires needed to build the EBN can be halved. The term *bidirectional* as used here refers to half-duplex wires, which can be driven in only one direction at a time. A protocol must be enforced in order to ensure that data is not lost by driving both ends of a wire at the same time. Since there is no explicit hardware on the network which arbitrates mastership of the bidirectional channel, the lines are driven by open collector outputs in the nodes along with passive pull-ups, avoiding the physical hazard of having two active drivers of opposite sense on the line. The logical hazard of two messages driven onto the same line is only a problem if the node is in a state in which a number of different messages are possible. The problem is easy to detect. A router will not initiate transmission across a link that is already in use, therefore, any collision that occurs will involve in-phase messages. A router that has a message to propagate that is of higher priority than the message it is currently in the process of receiving will wait until the message is fully received and then will utilize the cycles that immediately follow to reply with its own message.

Because the EBN connections to the node are I/O ports, the node can monitor its own outgoing transmission. If the received transmission does not shadow what was transmitted, then the node concludes that the other side of the link is transmitting in synch and a collision is detected. While it is possible that the message from one side of the link will be uncorrupted, it is not possible for both sides to see uncorrupted messages from colliding messages except in the trivial case in which the messages are the same. At the end of the transmission, there will be a dedicated cycle in which either side can pull down the line as a sign that there has been a collision. Both sides will see this signal and initiate a protocol that forces and alternates mastership of the link to one end at a time. This temporarily creates a virtual unidirectional link which allows for the resolution of link ownership conflicts.

The choice of using bidirectional channels is orthogonal to the choice of using multiple wire channels or multiple bit messages. Any combination can be utilized. Because the message collision problem arises only when there is a choice of messages that can be sent, it is not a problem when single wire, single bit networks and protocols are used.

4.3.7 Performance

The Network Delay on the EBN is proportional to the diameter d of the network where d is defined as the longest minimum distance between any two nodes in the network, *taking faulty components into account*. For single bit networks, broadcast is bounded by ds, where s is the delay incurred during each hop that a message takes. For multiple bit or multiple wire networks where collisions must be arbitrated, maximum broadcast delay is on the order of $2ds$.

The effect of multiple bit messages on performance is simply to multiply s by a constant factor proportional to the number of bits in a message. The granularity introduced by the need to wait for the complete transmittal of the current message before a new, higher priority message can be propagated provides a constant cost term. Therefore, neither of these effects will cause more than a constant factor change in performance.

The value of d is variable, depending upon network configuration. For any network, the greatest possible value of d is N, the number of nodes in the network. However, most networks have a much smaller diameter. For example, an n-dimensional mesh with no faulty components has a diameter $d=nN^{1/n}$. When components fail and are configured out of the network, the diameter may grow. However, because the EBN sends messages on all possible paths from source to destination, the diameter stays near the minimum for small numbers of faults.

To gauge the effect of faults on the diameter of a network, the following experiment was performed: Randomly-placed link faults were injected into a two-dimensional mesh network. After each new fault was injected, the network diameter was determined. The results show that the network diameter initially tends to increase as minimum length paths between nodes become blocked. However, as the number of faults increases, the point is reached where the next fault cuts the path that determines d and disconnects the two nodes connected by that path. The diameter of the resulting subnetworks is less than the network diameter prior to the last injected fault. The results are not monotonic from that point and the diameter may go up or down with each new injected fault. Eventually, though, the diameter begins to shrink, dropping to a value of zero when all links are faulted and the mesh is totally disconnected. A plot of the network diameter versus the number of faults injected from one experimental run is shown in Figure 5.

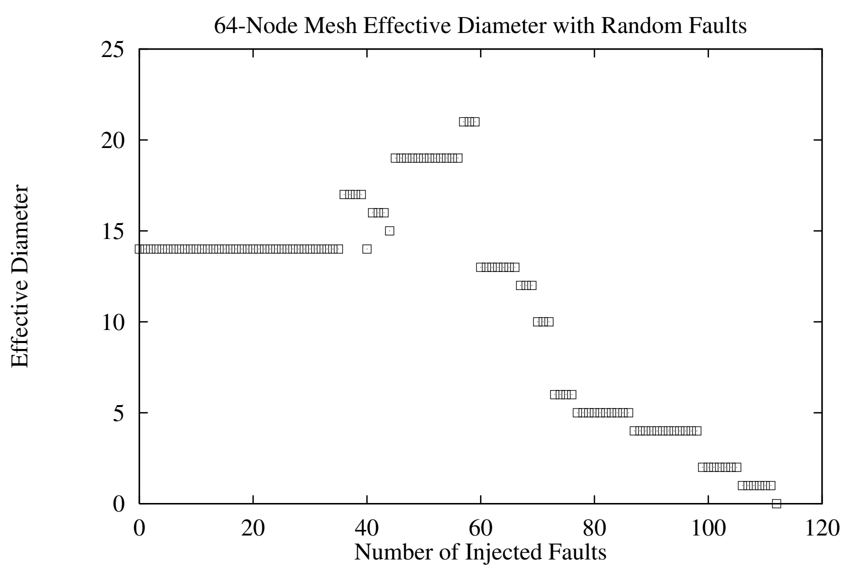


Figure 5: Network diameter vs. number of faulty links. This experiment is for an 8 x 8 node mesh network.

Statistics related to the maximum network diameter achieved in a series of experimental runs of this type are tabulated in Table 3. The results show that the maximum network diameter for two-dimensional meshes under the condition of random faults is roughly proportional to $N^{1/2}$, the length of the mesh in each dimension. This is significant because of the need to delay state transitions for a period proportional to the network diameter. The required delay will therefore grow slowly relative to N. Based on the experimental results, it appears that assuming a maximum network diameter of $5N^{1/2}$ gives a sufficient worst case margin. The experiment was repeated, faulting routers instead of links. This is the equivalent of injecting four simultaneous and correlated link faults. The maximum network diameter achieved was 15% to 30% less than for link faults due to earlier disconnection of the network.

Table 3: Maximum diameter of two-dimensional mesh networks with random faults. The mean maximum diameter, $\bar{d}_{max}$, and standard deviation, σ, are reported for 100 trials. The $\bar{d}_{max} + 3\sigma$ column indicates the number that bounds the actual maximum 99.8% of the time.

Number of Nodes N	Mean Max Diameter $\bar{d}_{max}$	Standard Deviation σ	$\bar{d}_{max} + 3\sigma$	$(\bar{d}_{max} + 3\sigma)/N^{1/2}$
16	8.9	1.5	13.4	3.4
64	21.8	3.5	32.2	4.0
100	28.9	4.7	43.0	4.3
144	35.8	5.4	52.1	4.3
196	42.8	6.7	63.0	4.5
256	49.7	7.0	70.8	4.4
400	61.9	8.4	87.0	4.3
576	76.5	11.9	112.3	4.7
784	92.8	14.9	137.6	4.9
1024	109.5	14.9	154.1	4.8

Consideration must be made of the potential effects of correlated faults. As shown by the experiment which faulted nodes instead of links, simply applying faults in close proximity will not cause an increase in network diameter beyond that observed under random injection. Even if the faults are carefully selected, it takes a significant number to increase the diameter of the network to $5N^{1/2}$. For a fault to cause the network diameter to increase, it has to force the minimum path between two nodes to backtrack (make negative progress). Intuitively, it would take more than $3N^{1/2}$ *directed*, not random, faults to increase the network diameter to $5N^{1/2}$. An example of this type of fault pattern which creates a snake pattern across the network is shown in Figure 6 for an 8 x 8 mesh network. In this example it requires 27 faults to increase the diameter to 40. For the case of a general mesh network with side $N^{1/2}$ and following this pattern of faults, it would take $7/2N^{1/2}$-1 faults to increase the diameter to $5N^{1/2}$. Any problem that could generate such a set of correlated faults would have to be global, not local. The expectation for systems utilizing the EBN is that periodic maintenance and repairs will be performed and the fault rate will be such that large numbers of faults cannot accumulate.

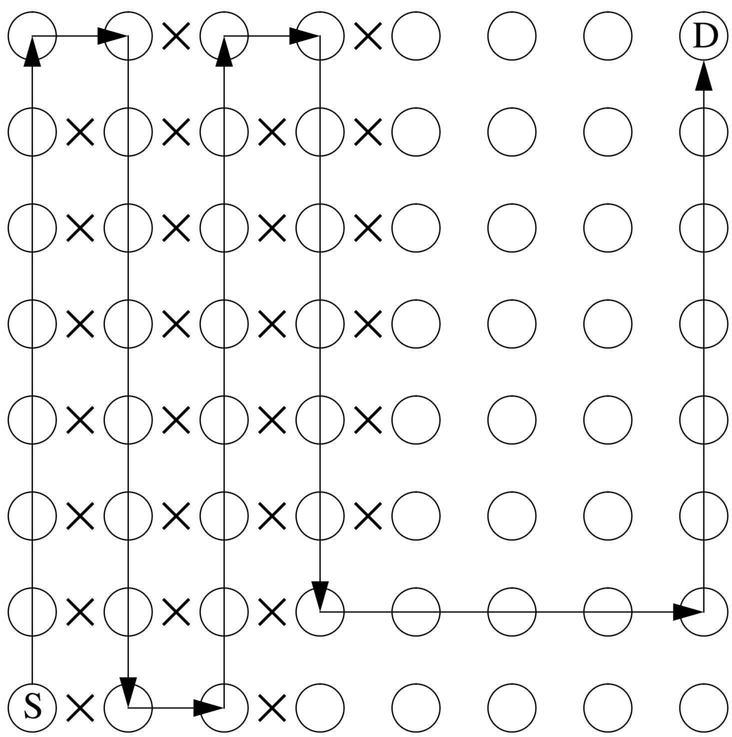


Figure 6: Network diameter increase due to faults. A message originating in source node S must make $5 \times 8 = 40$ hops to reach destination node D. The arrow defines a minimal path. X's mark faulty links. In this example it takes *27* faults to increase the network diameter from 14 ($= 2N^{1/2}-2$) to 40 ($= 5N^{1/2}$).

4.3.8 Synchronous/Asynchronous Behavior

The EBN does not require that the end nodes of each link be synchronized, but synchronous behavior will simplify the protocol. It is necessary that each node have a means to count the passage of time because of the need to delay state transitions for a period proportional to the end-to-end delay across the network. In an asynchronous system, timed state transitions by the router state machine controller will have to be made robust, possibly through the use of a synchronizing signal on the EBN. Asynchronous behavior will require more complicated protocols on the links that will allow for internal synchronization of a received message by the EBN node. The difficulty of implementing an asynchronous protocol on bidirectional lines will probably limit the usefulness of that implementation option.

4.4 Applications

The Express Broadcast Network has a wide variety of potential applications. A number of applications in which the EBN can be used to facilitate system startup, global flow control, fast barriers, and general control for networks are described. Section 4.5 describes the actual use of the EBN in the fault tolerant Chaos router.

4.4.1 System Startup

When a parallel computer system is initiated, one of the necessary functions involves the dissemination of configuration and partitioning control information by a master node. This is a situation in which the one-to-all broadcast mode could greatly simplify the problem. A master node can simply broadcast the configuration information to all nodes. This requires the use of one of the multiple bit transmission protocols described in Sections 4.3.4 and 4.3.5.

4.4.2 Global Flow Control

When traffic gets heavy, packet latency rises and at some level of traffic, throughput can drop. The data network is too slow for effective global flow control and any attempt to use it for such a purpose as the traffic load rises will only contribute to the traffic problem. The EBN can be used to provide effective fine-grained flow control for the network. Assertion of an EBN signal can slow or stop message injection. The flow-control algorithm itself is fairly arbitrary. For example, at some predetermined load level, a barrier could be initiated. In a non-minimal router, the need to deroute could be a trigger. An alternative trigger would be some level of utilization of router packet buffering. A router receiving a flow-control message could slow down or stop packet injection. The perceived load at the router receiving the flow control message could be a factor into the decision to stop injection. The flow-control barrier could have a timeout so that it would expire after a set period or it could take an explicit command to lift the barrier. The EBN simply provides a rapid and efficient means of implementation of the selected algorithm.

4.4.3 Application Barriers and Eurekas

The barrier and eureka functions of the EBN could be made available to software applications in order to expedite their function. Software barriers can be used to provide many useful functions involving synchronization. Eurekas can be used when parallel processes are attempting to solve the same problem using different mechanisms: the first process to complete signals all of the others by means of a eureka broadcast.

4.4.4 Control

The selection of a master node for control of certain functions, such as described previously with respect to system startup, could be simplified through use of the EBN. Each node is assigned a unique priority level. When the network moves to a state which requires that a master node be selected, each router listens for another router to ping on the EBN to announce that it has taken mastership. A fixed period is designated during which the highest priority router pings on the EBN if it is present and active. If the period passes without a ping, then a new period commences during which the next highest priority node pings. This continues until an active node is found to assume mastership. The fault tolerant nature of the EBN ensures that all connected nodes receive the highest priority ping, selecting a master in a consistent manner. If the network has become separated by faults, each separate connected block of components selects its own master by means of this algorithm.

4.5 Control and Synchronization of the Fault Tolerant Chaos Router

The EBN had its genesis in the need by the fault tolerant Chaos router for a broadcast mechanism that combines efficiency and fault tolerance with low cost. Given the goals of the fault tolerant Chaos design, system performance had to be preserved along with hardware resources and, therefore, network overhead dedicated to synchronization and control of fault management procedures had to be severely limited. The EBN allows this goal to be met. The Chaos router uses the the basic one wire EBN to implement a broadcast network for the control of fault management procedures. For the targeted design, the one wire EBN has sufficient functionality. Had the enhancement of general purpose network control also been considered for this design, the EBN functionality required for fault tolerance could have been smoothly folded into an extended EBN design of more general utility.

In this section, a general description of the flow of fault management activity within the fault tolerant Chaos router is provided. Figure 7 shows the state transition diagram associated with the fault management procedures. The sections that follow describe the specifics of the state transition diagram and the role of the EBN in providing global synchronization for these procedures. The router state machine sequence depends upon control from the processing node along with input from the EBN which provides global context. A detailed description of the mechanisms involved in the fault management activity is included in Section 5.

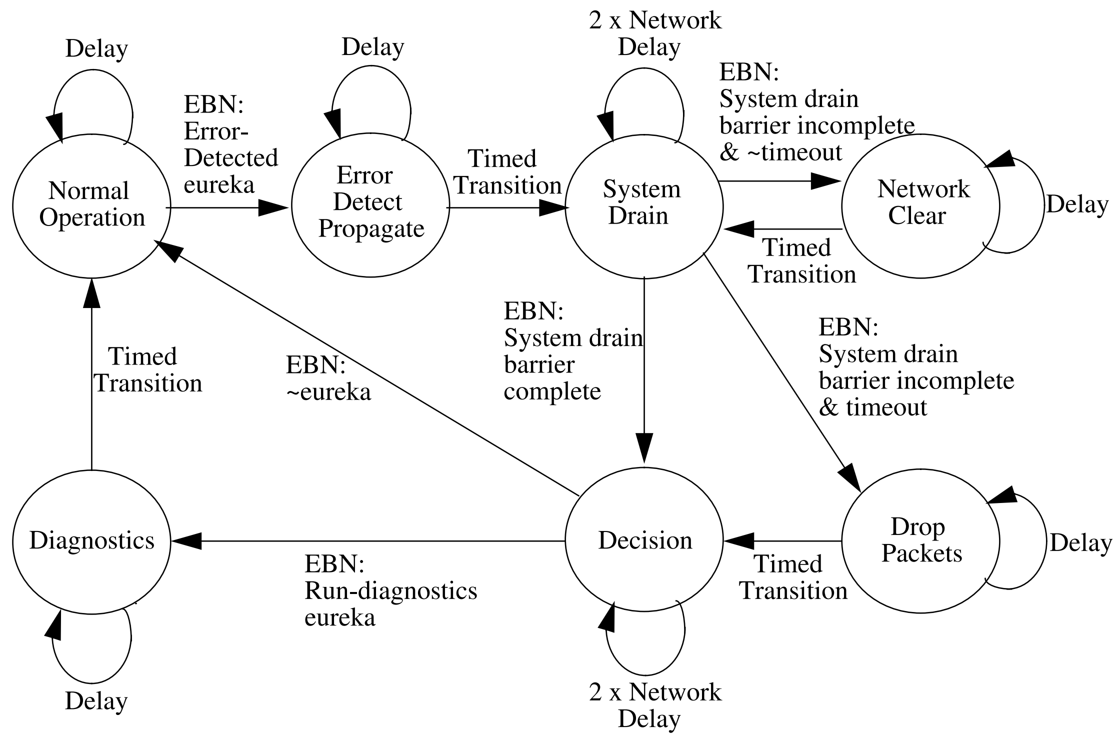


Figure 7: Fault tolerant Chaos router state transition table. The role of the EBN in fault management procedures is illustrated.

4.5.1 Chaos Overview

During normal router operation, the router state machine is idle. However, fault detection mechanisms are active on every cycle. Any node that detects an error generates a network broadcast which kicks each receiving node into the system drain procedure. Packet injection is inhibited and the network is emptied. If any error is still outstanding at this point, the network moves into a round of diagnostic tests. Otherwise, with no error pending, the network returns to normal operation. When diagnostics are completed, the network is reconfigured to remove faulty components from use. Finally, fault management terminates and normal operation is resumed. The Chaos router utilizes both types of EBN communication primitives, eurekas and barriers, to control the router state machine and the fault management procedures. The router also demonstrates the various types of state transitions that can be generated using the EBN: timed with the selection of transitions determined by the EBN input, timed with a predetermined transition (no EBN input), and untimed, triggered by the EBN.

4.5.2 Error Detection

The fault management procedures are idle while the router is in the *Normal Operation* state. The router state machine maintains the Normal Operation state until an *error-detected* eureka broadcast is received on the EBN. Any node that detects an error can initiate the eureka. The error-detected eureka is similar in function to the *global faulting* mechanism in the Vulcan architecture [Stunkel et al. 94]. The eureka forces an untimed transition into the *Error Detect Propagate* state that implements a short delay. The delay is needed to ensure that any ensuing EBN traffic trails the broadcast wavefront created by the error-detected eureka signal. Initiation of the error-detected eureka comes from the processing node which drives the EBN to its associated router. The router has error detection capability, but detection of an error by the router results in a private communication on the router-processing node channel. The processing node maintains final control over the initiation of the error-detected eureka broadcast. From the Error Detect Propagate state, a predetermined and timed transition into the *System Drain* state is made when the delay expires.

4.5.3 System Drain

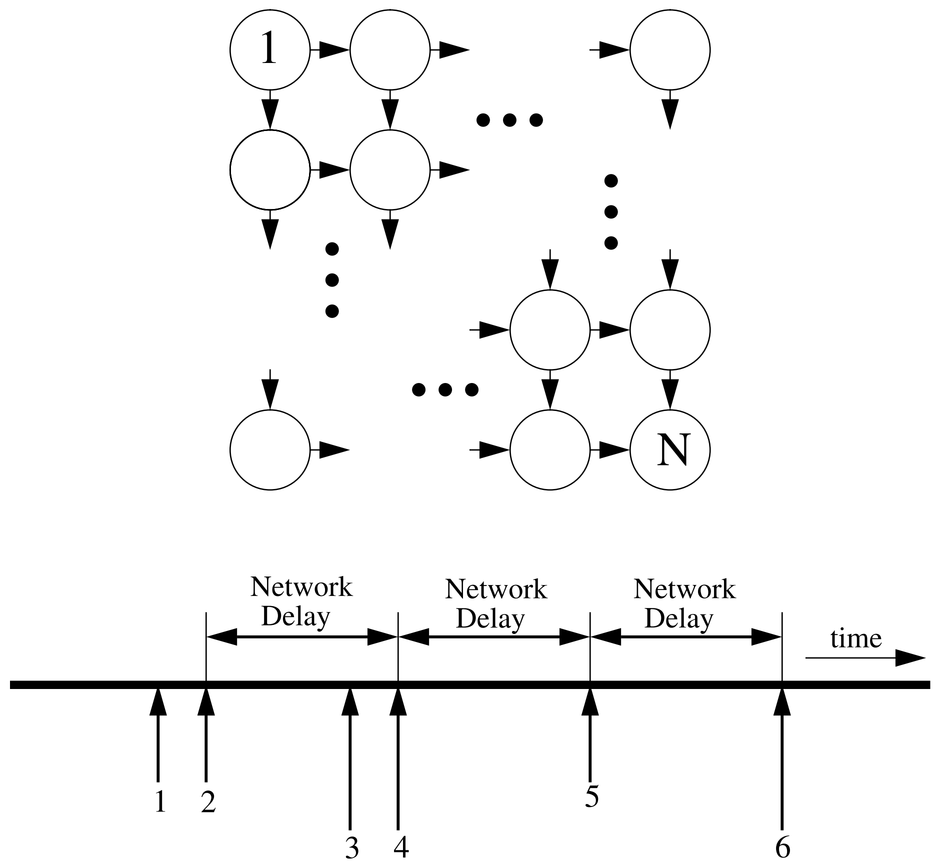
In general, the system drain ensures an upper bound on the transit time of a packet through the network. As a fault management procedure, the system drain is used to flush out any “wandering” packets and to set a known state into the network. This stage of the fault management procedure empties the routing network in order to create uniform empty and idle conditions across the network in preparation for diagnostic procedures. Injection of new packets into the data routing network is disabled, however, routing of packets already in the network continues. Preferably, the network will empty without intervention through normal packet delivery but packets will be removed from the network by direct action if necessary.

The EBN barrier function is used to detect the completion of the system drain. Implementation of the barrier requires a pair of closely linked states. These states cycle in a repeating sequence that first tests for the condition that allows the loop to terminate, then clears the network in preparation for a subsequent repeat of the test if the condition has not been met.

In the System Drain state, a router asserts the EBN if a packet header is present in the router. The conditional test, or barrier condition, is that the barrier is not completed until all packets are removed from the network. At the end of two Network Delays, the router checks the barrier condition and moves into the *Network Clear* state. This is a situation (see section 4.3.3) in which the combination of phase delay plus broadcast delay makes an extended wait necessary in order to maintain network synchronization. Two Network Delays are required because the sole packet in the network may be resident in a router that entered the System Drain state one Network Delay after the first router that did so. This scenario is illustrated in Figure 8. A router that has a packet header present at any time while in the System Drain state will assert the EBN for the remainder of the delay period in order to guarantee that the EBN assertion will be visible and uniform to all routers when the delay expires and the condition is checked. Lack of assertion of the EBN at that time, is an indication that the barrier is completed and the system is drained.

The Network Clear state prepares the network for the next test of the barrier condition, The router deasserts and disables the EBN for twice the delay incurred by each hop from router to router. One such delay is the time it takes for neighboring routers with a trailing phase to also move into the Network Clear state. It then takes those neighboring routers one more such delay to halt assertion of the EBN to all of of their neighbors including those routers with a leading phase. By temporarily disabling the EBN in this manner, it is ensured that when a router reenables the EBN, there will be no neighboring routers still asserting the EBN as part of the previous test of the barrier condition. The procedures performed in the System Drain state and the Network Clear state together implement the barrier function.

A timeout period is used to handle the situation in which the network contains undeliverable packets. If the system drain barrier is not completed after a period of time such that with high probability all deliverable packets have been delivered, then the timeout kicks the network into the *Drop Packets* state. Here, all packets that are not stuck are delivered to their current node. Any remaining packets are then noted and dropped. The transition out of the Drop Packets state is a timed transition, with enough time allowed for a router that has full buffers to deliver or drop all packets.



1) Router 1 asserts EBN (error-detected eureka), enters Error Detect Propagate state.
2) Router 1 enters System Drain state.
3) Router N receives error-detected eureka, enters Error Detect Propagate state.
4) Router N enters System Drain state, asserts EBN (packet in router).
5) Router 1 receives EBN barrier message from Router N. Router 1 decision: Is system drained?
6) Router N decision: Is system drained?

Figure 8: System drain timing and delays.

4.5.4 Diagnostics

Once the network is emptied, either through the graceful completion of the system drain or by forcibly dropping packets, the *Decision* state is entered. To be decided is whether or not to move into the diagnostics mode. Once again, the EBN is used in the eureka mode; any node that deems it necessary to run the diagnostics asserts a eureka. Again, a two Network Delay wait is required to allow the router with the greatest phase delay sufficient time to propagate a eureka back to the router with the most advanced phasing. The eureka broadcast drives the system into the *Diagnostics* state. In this state, each node runs diagnostic tests which enable the processing node to conclude on the health of its connecting links and neighboring routers. Bad channels are marked as inactive by the processing node. The new configuration is reported to the system software which can force additional network reconfiguration if global problems with the new network configuration need to be addressed. A timed transition is made to the *Normal Operation* state in which another short delay is counted off to avoid any conflict on the EBN with any ensuing error-detected eureka signal.

5 Fault Tolerant Architecture

The fault tolerant Chaotic routing network requires close interaction between hardware and software. A layered, hierarchical approach is used in which the hardware feeds information to application software which processes the data and in turn, supplies the system software with a higher level abstraction of the state of the system. While most of the detailed hardware design for the purpose of fault tolerance is directed at the Chaos router, the network interface design is also critical in providing system fault tolerance. The following section states some of the guiding principles that direct the design of the fault tolerant Chaotic routing network. Following that is a detailed description of the mechanisms that are used to implement fault tolerance.

5.1 Design Principles

As described earlier, a key issue in the design of the fault tolerant Chaotic routing network is the need to minimize increases in the cost of the system while avoiding the wholesale use of scarce network resources such as pins, wires, router circuitry and network throughput. This has several important ramifications:

- Specific points in the system are targeted as locations at which additional circuitry can enhance fault tolerance. The targeted locations provide for significant coverage of faults but with only an incremental amount of additional hardware. The approach is to take the macroscopic view and detect the high level error syndromes rather than the microscopic view which would place detection mechanisms in closer proximity to the fault locations. The general approach of using system-wide redundancy, which would provide for greater fault coverage, was rejected because of its cost.

- Since the amount of available resources is limited, a fundamental principle of computer design, “optimize the common case”, guided this design. Fault detection, the mechanisms of which are active during every cycle, is implemented in hardware. Hardware is also used to implement the enhancements to the routing algorithm which improve routing performance in a degraded network, another operation that may need to be performed every cycle. The fault management procedures, in particular the diagnostics and network reconfiguration, are rare events and are left to software.

- Software was rejected as the mechanism for the implementation of fault tolerance. It tends to use a significant fraction of system processing power and network bandwidth while not providing the fine granularity of detection that the hardware architecture can provide. A comprehensive implementation in software such as SIFT [Wensley et al. 78] provides for the redundant execution of tasks in separate processors with voted results. This approach can be effective but it goes well beyond the scope of the network fault tolerance that is the focus of this study. Smaller scale utilization of software methods, such as a reliable end-to-end data link transmission protocol in which packets are acknowledged by the recipient, are more relevant to this discussion. The use of such a protocol can cause the network traffic to double and, while providing for reliable message passing, does not obviate the need for additional means of fault management in the form of diagnostic procedures and mechanisms for network reconfiguration. However, the use of reliable data link transmission protocols is orthogonal to the fault tolerant architecture and can be implemented on a case-by-case basis to provide for additional reliability as needed.

•The architecture does not attempt to maintain the original network configuration through the use of spares to replace failed components. As faults accumulate, the network and computer gracefully degrade as functionality is lost. Functions that had been assigned to faulty or blocked nodes must be distributed among the remaining healthy nodes.

•Policy decisions are deferred to software control rather than built into the router. This removes complexity from the hardware and maintains flexibility in the system. For example, fault detection by the router results in a private communication to the processing node which, in turn, chooses whether or not to assert an error-detected eureka signal.

5.2 Implementation

The major issues in the implementation of the fault tolerant architecture are communication and synchronization, fault detection, diagnostics, reconfiguration and routing algorithm modifications to account for failed components. Communication and synchronization were discussed in Section 4. A discussion of the other topics follows.

5.2.1 Fault Detection

Perfect zero-latency fault detection would require duplication of all circuitry with a comparison of each node value. Besides being exorbitantly expensive, such close coupling leaves the circuitry vulnerable to common-mode errors. A better solution performs higher level comparisons of block and chip outputs. This provides near-complete fault coverage but still requires more than 100% overhead in circuitry. The point-detection approach used in the fault tolerant Chaotic routing network provides very good coverage while minimizing the cost of additional circuitry. While the number and variety of faults that may occur is enormous, there are a limited number of ways in which a fault may be manifested as an error. Therefore, the number of fault detection mechanisms required is also limited.

A two-fold approach is used for fault detection. The first set of fault detection mechanisms take advantage of the fact that the links are chokepoints upon which all communication with neighboring routers must take place. Any failure within a node that causes an error to escape the node will have a good chance of being detected by a neighboring node that monitors the content and form of all communications. A detectable error on a data link will be manifested either as a packet whose content is corrupted or as a communication that does not follow acceptable protocol.

The second set of fault detection mechanisms detects failures that do not create explicit communication errors but are detectable through an examination of the larger context in which the network message stream exists. This type of failure is manifested by such problems as late or missing packets or misdelivered packets.

5.2.1.1 Data Error Checks

Packet data errors are easily detectable through use of error control coding methods. These methods provide redundancy for the data by coding data words into code words that contain more bits than strictly required to hold the information. The simplest example of error control coding comes from the addition of a single parity bit to each data word. This guarantees detection of a single bit error in the code word.

Through the use of additional parity bits, error detection can be extended to multiple erroneous bits and error correction can be implemented. Hamming codes are a well known example of error correcting codes. Chen describes a number of codes having extended error detection and correction capability [Chen & Hsiao 84]. A code can generally be tailored to provide the desired error detection and correction capability.

Cyclic redundancy checks (CRCs) are a commonly used form of error control coding. They are useful because they can detect all single errors in a code word, burst errors up to a given length and many other patterns of errors, depending upon the particular implementation of the code. Another convenient feature is that an encoder/decoder in the form of a linear-feedback shift register (LFSR) is quite simple to implement. This implementation uses serialized data making such coding feasible for end-to-end checks but not for on-the-fly checking by intermediate routers on the routing path.

An alternate method uses a checksum. The checksum for a block of s words is formed by summing modulo-n all s words with n arbitrary. Coverage increases with larger n to the point that the sum of the s words is guaranteed to be less than or equal to n. The combination of the checksum and the s data words forms a code word. The checksum is suited for applications in which data is transferred in blocks so it is a good fit for packet routing architectures. The checksum is efficiently implemented by feeding a stream of data operands into an adder/accumulator. This scheme is also appropriate for end-to-end checks.

A packet in the Chaos routing network is split between dynamic and static sections. The first flit (16 bits) of the header is divided into an X displacement field and a Y displacement field (for a two-dimensional router). The data in these fields are dynamic, with an increment or decrement operation performed at each hop to represent the progress that the packet is making through the network. The remaining flits in the header and all payload data flits that contain message content are static. Separate error detection mechanisms are used for the dynamic and static sections of the packet.

A packet with corrupted data in the X and Y displacement fields will end up being misrouted. This in itself is not a disaster because the header contains fields that store the source node and destination node identifiers. Only the displacement fields in the header are actually read by the router and the remaining header space is loosely defined in the basic Chaos router, depending upon the needs of the network interface. However, the fault tolerant architecture requires that the source node and destination node identifier fields be included in the header. Cranium, a network interface designed for use with Chaos [McKenzie et al. 94], specifically defines these header fields as a source of information used for error detection. Using these fields, the processing node to which the packet is misdelivered will be able to determine that the delivery is in error and it can choose to reinject the packet, send a message to the source node requesting retransmittal, assert an error-detected eureka or simply drop the packet. However, in the interest of minimizing fault latency and eliminating unproductive network traffic, the Chaos router incorporates a mechanism for the early detection of this type of error. Simple parity is used to protect the displacement fields. The parity bit can be generated with minimal delay using a four level XOR tree. If the router detects a parity error, the result will be the immediate delivery of the packet to the processing node. This is similar to the *exception eject* mechanism in the R2 router in which a packet is delivered to the current node when the actual destination appears to be unreachable [Davis et al. 94].

There is one bypass to this procedure. Since any packet is subject to being derouted, the packet with the parity error may end up being derouted to another node while waiting to be delivered. In this situation, the normal header update will take place as the packet is routed and new parity correctly corresponding to the updated but corrupt displacement information will be generated and saved in the packet. This eliminates the possibility of early detection by other routers which would force early delivery. However, given that the situation is pathological, this exception will have little effect on network traffic. Consideration was given to forcing an improper parity bit into the header as it was updated as a means of propagating the error such that the router at the next hop would be able to detect the error and have a chance to remove the packet from the network. However, since the adverse effects of allowing the error to remain temporarily latent are small, that option was not implemented.

An alternate approach is to prevent packets that are to be delivered to the current node from entering the multiqueue, thereby eliminating derouting as a possibility. This approach has negative performance implications because the channel associated with the input frame in which the packet resides will stall if there is any delay in gaining access to the delivery channel. However, it is a safe option. There are no dependencies to deadlock the router/processing node link so the stall will eventually terminate. A solution is to limit only those packets with parity errors in the header from the multiqueue. This maintains the efficient approach for the common case.

A different error detection mechanism is applied to static packet flits. A complication with any data error detection scheme is that the virtual cut-through feature in the Chaotic routing algorithm allows a packet to be spread across a number of routers at any given time. Even if error control coding is done on a per-flit basis, by the time a given flit is checked, earlier error-free flits in the packet may have already left the router and it will be impossible for the router to respond to the error. Because of this, error detection in the static flits is left to the destination processing node. The parity check on the dynamic displacement fields of the header does not face this problem because that check is only performed on the initial packet flit.

The checksum appears to be the cost-effective choice for the static packet flits. The checksum requires only one or two additional flits, depending upon the error coverage desired. As a packet is delivered, the stream of flits can be fed into the checksum generator. An equivalent data-flow implementation for a CRC would require that each bit position be fed into a separate CRC encoder/decoder. This scheme would stretch the packet length by one flit for each extra bit required by the CRC code. An alternate, off-line generation scheme for the CRC could be designed that would be less costly in flits but the advantage and simplicity of the on-line data-flow scheme would be lost.

The choice of coding method for the static flits is a parameter which depends upon the network interface implementation and the amount of protection desired from the code. It is possible that error correction coding (ECC) may become a viable choice in a harsh environment with a relatively high rate of data error generation. ECC provides for fault masking by enabling the recovery of the correct data word from a code word with a limited number of bits in error. The use of ECC would lower the data word error rate relative to the code word error rate, thereby decreasing the amount of intervention required to assure reliable packet delivery. The use of ECC would require some additional hardware in the network interface to decode the correct data word from the received code word but the largest cost lies in the extra parity bits required to implement ECC which directly translates into a lower network data bandwidth.

5.2.1.2 Channel Protocol Checks

Channel protocol is designed to adhere to three basic rules that govern channel ownership arbitration [Bolding 93]. The rules are designed to provide some “fairness” in establishing ownership of the channel and to force ownership to alternate. The three rules are:

1. A packet may be transmitted only if the receiving side has room available in its input frame.
2. If only the non-owning side can transmit a packet, ownership is yielded to that side.
3. One side may transmit two consecutive packets only if the other side has no transmittable packets.

The problems involved in designing an effective channel protocol checker derive from the fact that much of the information that drives the protocol is known only to one side of the transaction. For example, the state of the input and output frames are known only to the router in which they reside. Therefore, if a router fails to yield channel ownership, it may be due to a fault or it could simply be that the input frame of the channel owner is not available to receive a new packet. Most control sequences turn out to fall within the space of possible legal behavior. Thus, fault detection based on channel protocol checks is limited to recognition of a small number of unambiguously faulty control sequences along with several “sanity” checks.

In the channel protocol, the router which does not own the channel signals to the owner 1) whether or not it wants the channel and 2) whether or not it has an input frame available. If the non-owner wants the channel, then its output frame must be full. The only way for that to change is if a transmission across a channel empties the output frame. If the router has an input frame available, the only way for it to become unavailable is for it to be filled by a transmission across the channel. An error is indicated if the nonowner changes its signals to indicate that it no longer wants the channel or that it no longer has an input frame available, without the event of an appropriate transmission across the channel.

At the end of a packet transmission, the router that is channel owner signals to the non-owner 1)the end-of-message signal (EOM), 2) whether or not the router is giving up ownership of the channel and 3) if the router is keeping ownership, whether or not the router is going to transmit a new packet. An error is detected if the channel owner attempts to send a new packet to a router that does not have an available input frame.

The sanity checks are related to the amount of time that an output frame must wait until granted channel ownership and to the length of the transmission. There is a bounded wait for an output frame to take control of a channel because of the three rules above in combination with a limit on the amount of time that a packet may stay in an input frame before it is sent to the multiqueue. This bound gives force to the rules above. If a packet could legitimately become blocked in an input frame, that router could legally retain ownership of the channel in question for an indefinite length of time. By rule 1, the router could not receive packets on that link thereby rendering nontransmittable any packets on the other end of the link. Therefore, by rule 3, the router could retain ownership and simply continue transmitting new packets. However, due to the bounded wait, either channel ownership will switch from time to time or a violation of channel protocol will be detected.

The sanity check involving packet length is necessary because the end of a packet transmission is marked only by the assertion of the EOM signal. Otherwise, the transfer of a flit of data is assumed each clock cycle. If the EOM is faulted or affected by a fault such that the receiver never sees the signal, then without some limiting mechanism, the receiver could generate a packet of potentially unbounded length. Because the Chaos router assumes a maximum length packet of 20 flits, a problem of this nature can be detected by counting flits in each packet reception. If a packet does not terminate after the twentieth flit, the router will terminate the transmission and flag an error to the processing node.

The description of the channel protocol checks to this point has had an implicit assumption that data will be asserted on the channel to match the control signals. This means, for example, that in the case in which a router erroneously maintains ownership of a channel, it will simply continue to transmit packets out that channel. However, in the the presence of faults, data assertion may not match what the controls indicate. The two sides of the channel may not be synchronized as to the state of the channel and situations can arise in which both sides transmit or receive. Out-of-phase receptions can occur in which undriven flits are padded onto the head or tail of a packet and real flits are missed. This type of problem would generally corrupt the packet data and should be detected by a data error detection mechanism even if the channel protocol checks do not detect the situation.

In general, control failures that change the form of the channel communication will result in corrupted packets which are easily detected. A detection problem occurs when improper control does not result in corrupted packets but instead either delays or blocks the smooth transmission of packets across a channel. The sanity checks provide for a high-level fallback position which will detect most channel failures of this type. The context-type checks may also be able to detect problems of this nature. Ultimately, what will not be detected is "malicious" behavior in which fault detection is not triggered but the channel activity is slowed to the point that performance is hindered. While it is desirable to eliminate as many "latent" faults as possible from the network, this situation will not necessarily be less favorable than the response to an identified faulty channel which would be to shut the channel down entirely. Ultimately, such malicious behavior is likely to be caught if and when diagnostics are entered for other reasons. The highly regimented diagnostic phase puts specific and rigorous demands upon the channel activity.

While overall error coverage is acceptable with the limited amount of fault detection related to channel protocol, there are improvements that can make the channels more robust or minimize error detection latency. The basic design for the Chaos router channel architecture is as follows: All channel control is multiplexed onto 4 signal lines per channel. Two belong to the channel owner and two to the receiver. The lines are bidirectional and control over them is switched whenever the channel owner is switched. It is possible that due to faulty control, both routers on a channel may assert a given control line. Though this is clearly faulty behavior, it will not necessarily be detected because any conflict in signal levels will be an analog effect which the digital logic may not notice.

The simplest way to improve coverage is to replace the bidirectional channel control lines with two sets of unidirectional lines. This is done by statically associating a pair of lines with each router instead of dynamically associating a pair of lines with the channel owner and a pair with the nonowner. This eliminates the scenario that has both sides driving the same line at once and prevents the possible subsequent masking of failures. Similarly, by using two unidirectional sets of lines for the data instead of a single set of bidirectional data lines, the channels are made less susceptible to faults. However, this has added costs due to the need for additional control lines. This cost is multiplied by the number of channels into the router making this an expensive change.

Increasing the number of control signals in each channel could provide additional information that could be used to disambiguate certain situations that could be indicative of faulty behavior. Passing the state of the input and output frames of a router across a channel to the other router would provide some redundancy and context to the other control signals. However, as previously stated, channel expansion is a costly choice.

5.2.1.3 System Drain

The system drain provides for fault detection of the second type in which the message context is required in order to determine that an error has occurred. In fault management procedures, the system drain has two general purposes: to save user messages before beginning diagnostics, and to set up the network in a known state for the diagnostics that follow. The system drain is also used to confirm suspicions of missing packets.

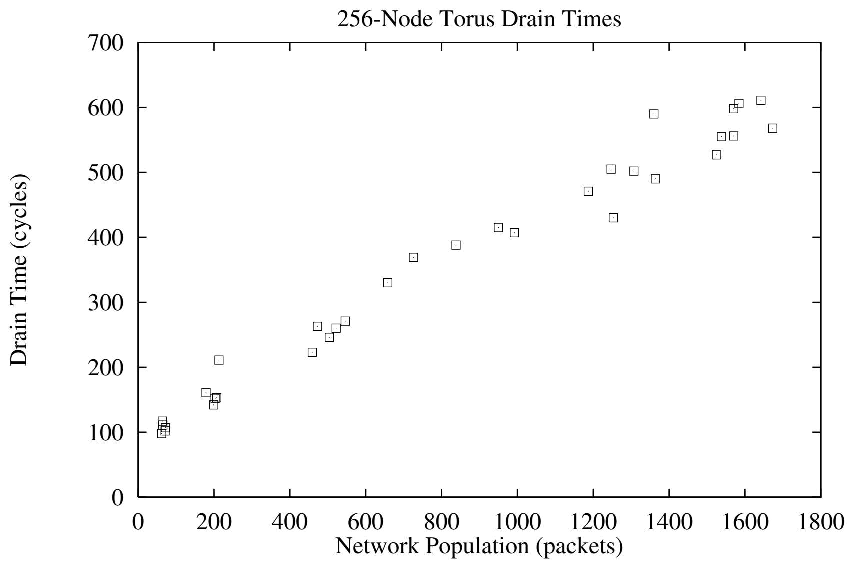
The need for such an error detection mechanism arises because of the adaptive nature of the Chaotic routing algorithm. Because packets that belong to the same message will not necessarily take the same path across the network and may be delivered out of order, it is necessary that the packets be reordered in proper sequence before the message is delivered to the destination processor. This function is performed by the network interface. All packets contain a message and sequence number which identifies their relationship to a larger message. The general procedure has the network interface placing delivered packets that are part of a larger message into a reordering buffer. When all holes in the sequence of packets are filled, the message can be delivered. Given that packet delivery in a Chaotic routing network is not guaranteed to occur within any fixed amount of time, it is possible that a missing packet simply took a longer or slower path through the network than did its fellow packets. The other possibility is that the packet in question was dropped, misdelivered or is stuck behind some obstacle created by faults in the network.

There needs to be a limit such that the network interface does not have an unbounded wait to determine whether the missing packet will ever be delivered. One possibility is to clear the portions of the reordering buffer that have an intact message segment. A hole in the sequence prevents any packets later in the sequence from being cleared. An overflow of the buffer indicates that a packet is late well beyond the network latency of the other packets in the message. Another mechanism used in Cranium [McKenzie et al. 94] is a watchdog timer which signals an alarm if the entire message has not been delivered within a fixed period after transmission begins. This protocol requires that there be a fixed maximum length for multipacket messages. The detection of a missing packet can be used to trigger an error-detected eureka signal which in turn leads to a system drain. Since the drain results in the delivery of all deliverable packets in the network, all missing packets that are simply late will be delivered. Because packets are injected into the network in sequential order, all holes in the reordering buffer should be filled when the system drain terminates in the absence of faults. An unfilled hole confirms that a packet has been lost. If the system drain results in the delivery of all missing packets and no other errors are pending, then diagnostics can be bypassed and the network can return to normal operation.

The implementation of the system drain requires the addition of a number of utilities to the Chaos routing network. To initiate the system drain requires that packet injection from all processing nodes be disabled. This step doesn't require any extra hardware because the error-detected eureka signal that initiates the overall procedure is propagated by the EBN to the processing node which in turn suspends packet injection at the source for the duration of the system drain. The processing node stays synchronized with the router during these procedures, monitoring and driving the EBN as appropriate at synchronization points.

The system drain timeout, needed to guarantee termination of the system drain, requires simply that a counter be available to count cycles during the drain. The choice of a timeout period requires analysis of the situations that may be encountered. Experimental data relating system drain time versus the population of packets within the network shows that the expected drain time for a network under typical experimental load conditions should be well under 10,000 cycles. Figures 9 and 10 show system drain times for 256 node and 1024 node torus networks as a function of the number of packets in the network. The experimental results are close to linear in the number of packets.

The highest network population tested in these experiments gave an average population of between 7 and 8 packets per router. Linear extrapolation of the data to a completely full network (15 packets/router) gives drain time estimates of 4600 cycles for the 256 node network and 4700 cycles for the 1024 node network under hot-spot traffic conditions. The numbers end up very similar because of the way in which experimental parameters interact. The controlling constraint on drain time in both networks is the router/processing node channel bottleneck in the nodes that are defined as hot spots. The backup of packets trying to reach the hot spot processing node has to serially cross the bottleneck. In the 256 node



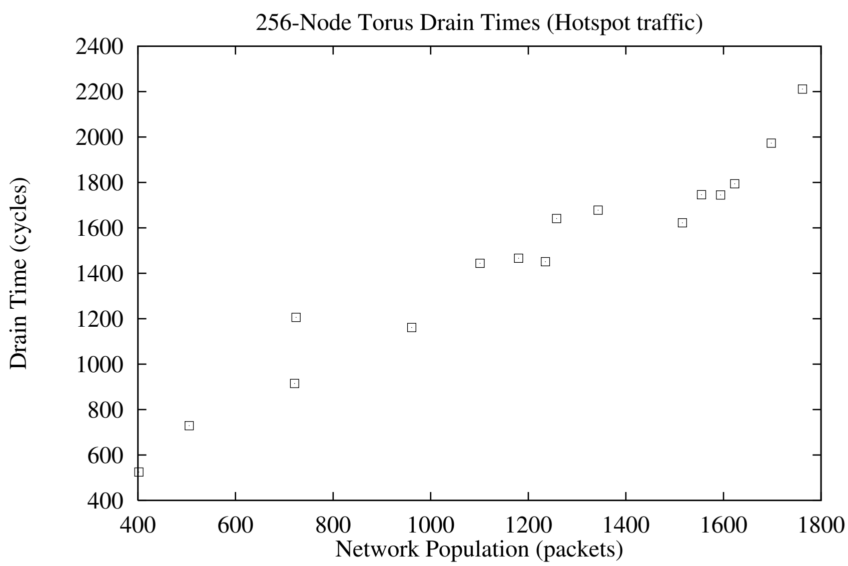
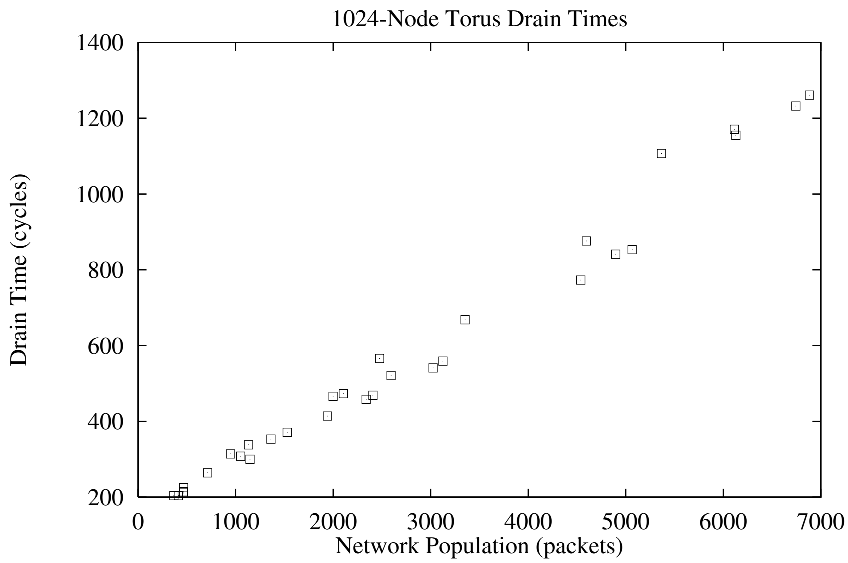


Figure 9: System drain times for a 16 x 16 node torus network. Packets are injected with uniform random destinations in the top figure, and with traffic containing ten ten-times hotspots in the lower figure.

network, the probability that a packet in the network has a hot spot node as its destination is 4x the probability of that in the 1024 node network because there are only 1/4 the number of nodes in the 256 node network. However, there are 4x the number of nodes in the 1024 node network to inject packets so in absolute terms, the number of packets backed up behind the controlling bottleneck is about the same in each case and matching drain times are observed.



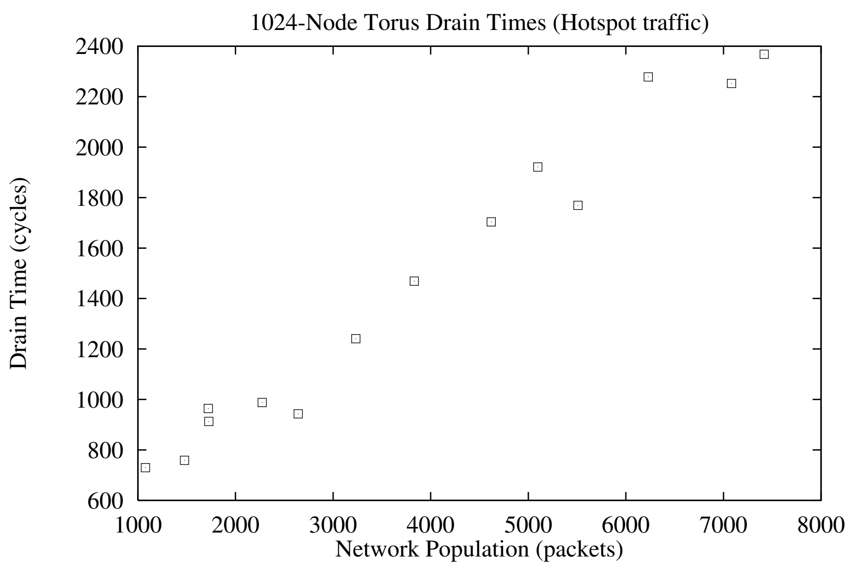


Figure 10: System drain times for a 32 x 32 node torus network. Packets are injected with uniform random destinations in the top figure, and with traffic containing ten ten-times hotspots in the lower figure.

An alternate analysis uses the Cranium network interface design to determine a bound on the size of the backup across the bottleneck link. Cranium requires that any multipacket message be set up by an exchange of control messages between source and destination before transmission of the message can begin. A limit of 16 such messages can be active at one time in a given network interface. Message length is limited to 2 Kbytes which translates into sixty four 20-flit packets (2 bytes/flit, 16 payload flits/packet). Therefore, the total number of packets in the network at any time associated with multipacket messages that have a specific node as destination is limited by 16 x 64 = 1024. Add 10% to account for single packet messages headed toward that destination node and another 10% to account for transmission overhead and inefficiency in setting up a continuous stream of packets across the router/processing node bottleneck and an estimate for the maximum system drain time is determined:

System drain time = 1024 packets x 1.1 x 1.1 x 20 cycles/packet = 24,781 cycles

An upper bound on the drain time can be calculated using the pathological case in which the network is entirely full and all packets have the same node as the destination. Given a 1024 node network with 15 buffers/router and 20 flit packets, the drain time is as follows:

System drain time = 1024 nodes x 15 packets/node x 20 cycles/packet = 307200 cycles

This calculation does not account for any inefficiency in setting up a continuous packet stream. The result is far greater than the more realistic estimates but even so, assuming a cycle time of 15ns, the drain only takes 4.6ms. This is not exorbitant assuming that the period between invocations is on the order of at least minutes. The drain time will also shrink linearly with cycle time allowing technology advances to improve performance. A realistic bound will depend upon the network interface implementation and the application load on the system but it should be at least an order of magnitude or more below the upper bound. Network performance will not be sensitive to the selection of a timeout period because normal system drain completion will in most cases preempt the drain timeout. Only in pathological cases or in the presence of a fault will the timeout be activated.

In a faulty network, the drain time may be unbounded. Consider the situation in Figure 11 in which faults make a network node unreachable. Any packets in the network that have that node as a destination will be undeliverable and the system drain will not be able to gracefully complete. For this type of case and for those pathological cases in which nodes may be deliverable but network load and traffic pattern prevent a full drain from completing within the timeout period, a means to clear packets from the network after the timeout expires is required.

The first phase of clearing the network is accomplished by delivering to the local processing node all packets received after the timeout. The mechanism is the same as that used for packets that have bad parity in their headers. The routing decision logic marks the packet as being at its destination node which results in its immediate delivery.

The second and final phase takes place after all deliverable packets from the first phase have

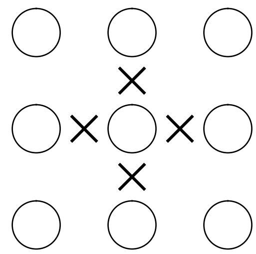


Figure 11: Example of unreachable node. The center node is unreachable due to faulty links which are indicated by the X's.

been delivered. At that point the only packets remaining in the network will be in the multiqueue or output frames, presumably blocked by a faulty component. The packet and the datapath do not need to be touched but the multiqueue scoreboard and the output frame control that indicates that a packet is present both need to be cleared. Any router which clears a packet in this manner will report that information to the processing node as evidence of a detected error.

Fault detection by means of the system drain depends upon a knowledge of the context of the network message stream. In the fault tolerant architecture, the network interface has an expectation that packets associated with multipacket messages will arrive with some degree of timeliness. However, this does not work for single packet messages in the network. From the network standpoint, they have no context and therefore late or missing packets of that type cannot be detected by the network or network interface. Actually, all messages of any type have a context but that is to the software process involved in the communication. Out of all single packet messages, some will not be significant and won't be missed, some may be expected by the applications with which they are to communicate and are therefore detectable if missing and some may be important but unscheduled and therefore not missed if not received. Though the desire to eliminate the overhead of a reliable data link transmission protocol is one motive for the design of a fault tolerant routing network, this architecture is not able to completely eliminate that need. The use of such a reliable protocol on a spot basis as a software overlay on the fault tolerant hardware architecture can provide added reliability.

An understanding of the distribution of message sizes in a multicomputer workload is needed to gauge the significance of the problem of context-free messages. In [Cypher et al. 93] the communication requirements of a number of scientific computing applications running on multicomputers was examined. It was shown that 48% of the messages passed were

16 bytes or less long. These messages would fit in a single packet. However, if the observed distribution of messages were to be packetized with 32 data bytes/packet, then less than 1% of all packets would be self-contained messages not associated with a larger packet stream. This indicates that the fault detection mechanisms associated with the system drain process and the network interface will provide high coverage for context-type errors.

The network interface has several additional error detection duties. A packet that is delivered to other than the desired destination node, either in error or as the result of a detected parity error in the header, will be detected by the network interface when the destination node field in the header is read.

The issue of detection of multiple copies of a packet generated in error is a more difficult problem. If the packets are part of a multipacket message, then extra copies can be detected by the reordering buffer. But, for single packet messages, the error is not covered. This is a pathological situation, however, because the Chaos network does not maintain duplicate copies as part of a reliable transmission protocol. It is far more difficult to postulate that extra copies somehow are generated than it is to assume a failure which fails to eliminate a duplicate copy in a system that creates such copies as a matter of course.

5.2.2 Diagnostics

Simple, local algorithms for diagnostic tests were selected both for the sake of efficiency and to avoid byzantine problems in which some form of distributed agreement must be reached among possibly faulty components.

5.2.2.1 Off-line Testing

Once the system is drained, the diagnostic procedures can proceed. The goal of these procedures is to identify as many different faults as possible using a quick distributed algorithm. The strategy is to exercise router function in a controlled manner such that much of the data path is tested allowing associated errors to be identified while counting on the channel protocol checks to catch and identify errors other than data errors in the routing process. The finite nature of the test and the limited number of injected packets during the procedure allow for guaranteed completion of the test within a fixed period. The procedure is straightforward and simple: All nodes inject test packets destined for each of their neighbors. When a node receives a test packet, it returns an acknowledgment message to the sender. This tests the path shown in Figure 12(a) in both directions. If the message is not returned, or is returned corrupted, it may be due to failure of the link, one or both routers, or the processing node to which the test packet was sent. 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 exercised by more than one packet, since each node is connected to more than one link, and all links along with a good portion of the datapath are tested. The controlled set of messages and the expected response establishes a context that allows identification of the faulty components.

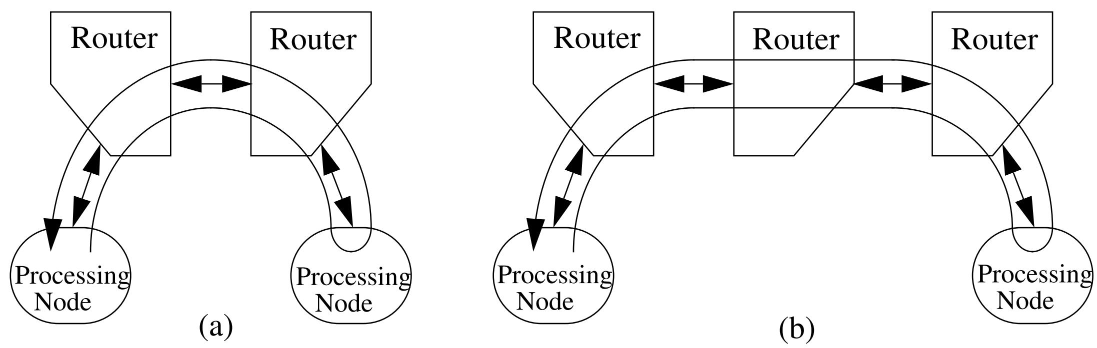


Figure 12: Fault diagnostic testing. (a) One-hop testing path, which requires nodes to be functional for the link to pass the test. (b) Two-hop testing path allows links to pass the test even if nodes fail.

While many faults will result in straightforward, unambiguous diagnoses, some errors may result in more byzantine diagnoses with apparently inconsistent results. For example, a link may be marked as bad on one side but good on the other. This situation will resolve itself quickly as a router will not communicate across a link that it has marked as unusable. This lack of communication will be detected by the neighbor router as a failure to maintain protocol with the eventual result that the neighbor router will also mark the link as unusable. A faulty router may be so marked by some of its neighbors but not by others. The fault may only affect some channels or it may have been latent in the diagnostic tests performed across some channels but not others. This is not unexpected and routers are expected to operate even if only partially functional. Network configurations in which obstacles of greater than simply local scope are generated can arise as a result of the distributed local reconfiguration carried out based on the diagnostic results. This situation is dealt with by system software as discussed in Section 5.2.3.

The basic diagnostic procedure can be extended to provide for greater datapath coverage. By extending the testing to two-hop paths as shown in Figure 12(b), more of the virtual cut-through router datapath can be exercised. This extension also allows for some flexibility in the reconfiguration decision. If the processing node is dead, then the single-hop test results will cause each of the neighbor nodes to mark the link to the dead node as unusable. This will render useless a router attached to a dead or missing processing node. The two-hop testing allows the router to remain active even with a dead or missing processing node. If a node sends a message to a neighboring node as part of the diagnostics and receives no answer but does receive an answer to a message sent over the same link as part of a two-hop test, then the node can mark the neighboring processing node as dead but the associated channel may remain as a routable direction. This approach is conservative of hardware and also maintains the usefulness of the fault tolerant architecture in a network that is by intent less than fully populated with processing nodes.

This procedure can 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 is also undesirable to allow extended paths through routers without active processing nodes. Because the diagnostics have a strict locality of scope, the number of such nodes with inactive processing nodes must be limited in order to maintain reliability. A conservative approach would prohibit active routers with inactive processors in adjacent nodes. This is a policy decision that does not require hardware support by the router.

Additional test extensions can be used to exercise the datapath through the multiqueue. Virtual cut-through can be disabled and packets received by input frames can be routed into the multiqueue. This will exercise the multiqueue fifos. While it cannot be guaranteed that all crossbar switches will be exercised, most paths should be covered.

There are complications with this test design. Packets are not derouted out of the injection node so packets that are part of a one-hop test will traverse only the link under test. However, in many cases, a packet that is a part of a two-hop test has more than one choice of directions by which it may be minimally routed. Therefore, the targeted link under test will not necessarily be traversed by that packet. A fix would be to flag the header displacement field in the test packet to indicate which dimension should be routed. Another fix would be to hardwire a sequence into the router such that when in diagnostic mode, the router would route the stream of injected packets in predetermined directions. However, both schemes would affect the critical path through the router by adding complexity to the header decode and routing decision logic. The second proposal also eliminates flexibility and some of the software control over policy by fixing in hardware what had been strictly a software function. Two-hop testing can still be performed without having to add logic to the routers if paths are only allowed in a single dimension. This guarantees that the first hop will be across the desired link while there are no constraints on the second hop. Minimal routing will take the packet directly to the destination but derouting is possible, increasing the length of the route. A return acknowledgement from the destination would still indicate that the target link had functioned, even if the second hop actually turned into multiple hops. This scheme does not achieve the datapath coverage that would be achieved with the full two-hop test but still provides some of the benefits and does not require any router modifications.

Fairness in access to the network is not an issue because the test is finite and enough time is allotted for all packet injection to take place. However, packets in output frames that do not progress across their link will be dropped in order to avoid blocking the progress of other injected test packets. The resulting lack of an acknowledgment for the dropped packet provides direct evidence of the identity of faulty components.

5.2.2.2 On-line Testing

The diagnostics described above are run only when the system has been drained and is in a special off-line diagnostic state, triggered by one of the fault detection mechanisms. In order to provide more timely fault detection, on-line diagnostics were considered. The nodes would inject test messages to each of their neighbors, as in the basic diagnostic procedure above, only during normal network operation. If no response were received within some specified time period, the node could generate an error-detected eureka signal and the network would eventually enter the off-line diagnostic procedures as needed. This idea, unfortunately, has all of the complications of the off-line testing and more. The major new problem has to do with indeterminate delivery time during on-line operation. This creates issues involving the determination of whether a message is late or lost which could limit the effectiveness of the test. Due to the complexity of the problem, on-line diagnostics do not appear to be a practical addition to the router.

5.2.2.3 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 guarantee coverage all of the possible message paths through the router, some 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 the fault detection scheme.

The health of the processing node is a key issue in the determination of the health of the network. The network diagnostics provide only a minimal test of the processing node function. Additional diagnostics for the processing node will be required for the sake of system reliability but that is beyond the scope of this design. However, the network configuration will be a vital input to processing node diagnostics. Built-in self-test is a possible approach but a more robust approach has neighboring nodes testing each other [Somani & Agarwal 92]. This process is similar to the network diagnostics but is far more comprehensive. The nodes apply test vectors to their neighboring nodes and then vote on the individual health of neighboring processing nodes. The network configuration map provides a baseline input as to the eligibility of nodes to participate in this process. In turn, the results of the processing node diagnostics will be applied to the network configuration map.

5.2.3 Reconfiguration and Routing Algorithm Modifications

Reconfiguration marks a network resource as no longer functional. The routing algorithm needs to incorporate this information into the routing decision such that only functional components are used to route packets and network obstacles created by failed components can be smoothly avoided.

5.2.3.1 Local Reconfiguration

The processing node controls the local configuration by means of a configuration register located in the router. The processing node can write a configuration vector to that register which indicates whether each channel is usable or faulty. The routing algorithm uses this information in the determination of possible routes. This capability is useful for flagging faults and for marking network nonuniformities. Using the example of a mesh, the edge channels that are unconnected must be marked as unusable so that the routing algorithm does not attempt to route packets in those directions. Control of the configuration vector allows the processing node to dynamically reconfigure components out of the system. However, reconfiguration is only done under controlled circumstances such as during system initialization and after fault diagnostics when the network is empty. Restricting reconfiguration in this manner avoids the problems associated with changing the router environment while routing decisions are in process.

When fault diagnosis is complete, each processing node uses the test results to determine the status of the channels connected to its router and writes that status to the router configuration register. The configuration register contents act as a functional channel mask which is used to eliminate dead channels from routing decisions. This is a simple task for the router hardware. When a new packet arrives at the router, a list of directions which profitably route the packet is generated. This list is dependent only upon the contents of the displacement field in the packet header. The list of profitable channels is then logically AND'ed with the functional channel mask to generate the list of profitable *and* functional channels for use by the routing decision logic. Figure 13(a) shows an example of how this list is generated.

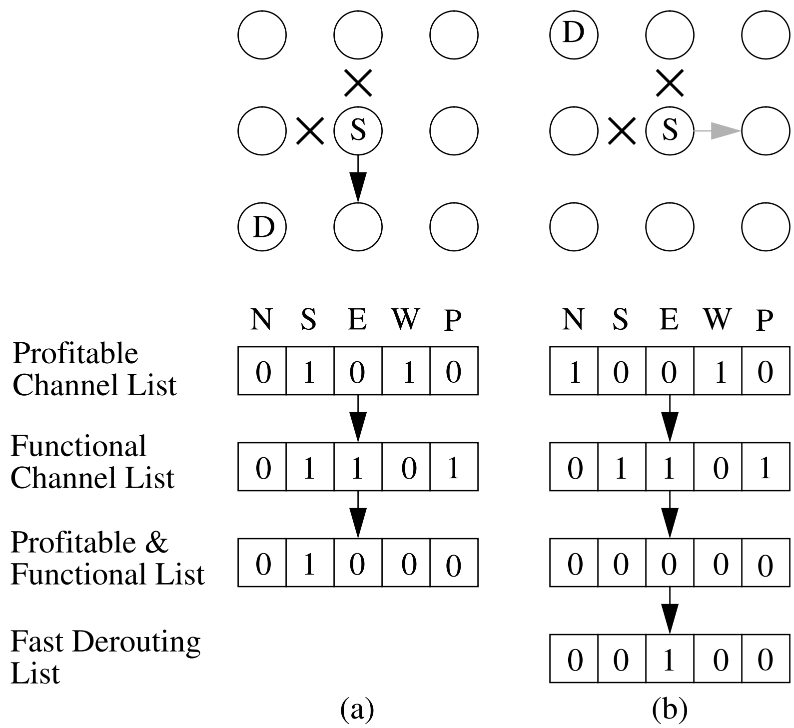


Figure 13: Use of the functional channel list in routing. Packet is currently in node S and needs to get to node D. Dark arrow signifies route. Lighter arrow signifies deroute. (a) Example in which profitable and functional route is available. (b) Example of fast derouting. Packet entered node S on South channel.

A *No Routeback* criterion is implemented to improve routing performance, particularly in the presence of faulty links. This criterion removes the channel on which the packet was received from the list of profitable and functional channels unless that channel is the only remaining functional channel. In a fault-free network, the only time this criteria will activate is when a packet is derouted in which case the return direction will be profitable but will not be entered onto the list of profitable and functional channels. In the Post Office router [Davis 92], this criterion is implemented when derouting as a means of breaking up cycles in heavily loaded networks in which a packet gets sent back and forth between two routers, eating up bandwidth. In the Chaos router, as will be shown, this criterion also turns out to improve routing performance in a network with dead channels.

If the packet's list of profitable and functional channels is empty, then the packet in the basic Chaos router will never cut-through to an output frame. It will be shunted to the multiqueue where it will wait until it is derouted out a nonprofitable channel. The fault tolerant Chaos router implements a utility called *fast derouting* that enables the packet to be derouted without accumulating delay as it would while waiting for an ordinary deroute. In fast derouting, the list of channels sent to the routing decision logic is set to the functional channel mask with the No Routeback criterion applied. The packet will not be delivered to the processing node using fast derouting. The logic to implement fast derouting is quite simple so it will have slight impact on the critical path for packets through the router. Fast derouting creates a short cut which avoids the normal mechanism by which packets are derouted around faults. Packets do not have to wait in the multiqueue to be randomly derouted in order to detour around faults and packets will not get stuck behind faults because, unlike ordinary derouting, fast derouting does *not* require traffic pressure to activate. Figure 13(b) shows an example of fast derouting.

The advantage that the No Routeback criterion provides in the presence of faults is shown in Figure 14. Dead links have created an obstacle that the basic Chaos routing algorithm will find almost impossible to bypass. A packet has reached node A on its way to destination node D. The packet will be derouted to node B or node C because there is no profitable route possible. In Figure 14(a) the No Routeback criterion is not in effect and from either of nodes B or C there is one profitable route, back to node A. In the absence of traffic pressure that can force another deroute from node B or node C beyond the obstacle, the packet will be livelocked, even with fast derouting. Next, consider the situation with the criterion in place as shown in Figure 14(b). From node B or node C there will be essentially a 50/50 chance that the next hop will take the packet to node E or node F respectively, from which profitable routes can be made that lead beyond the obstacle. The criterion gives a finite chance of escape from obstacles that extend linearly for even greater distances. If three deroutes are required to bypass the obstacle, then the second deroute will occur in the direction needed to bypass with probability 0.5 and the third will also occur with probability 0.5 for a total probability of 0.25 that the obstacle will be bypassed. Figure 15 shows this scenario. The same pattern is followed for larger obstacles.

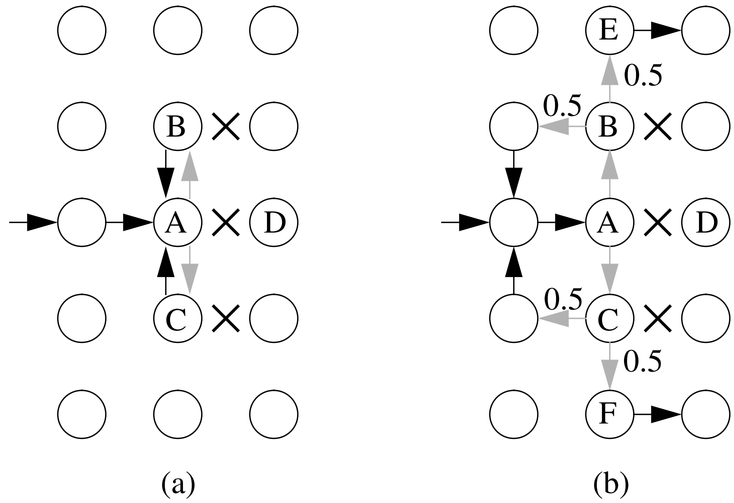


Figure 14: Bypass of linear obstacles. Dark arrow signifies route. Lighter arrow signifies deroute. Fraction associated with link denotes probability of deroute in that direction (a) Example in which No Routeback criteria is not implemented. (b) Example in which No Routeback criteria is implemented.

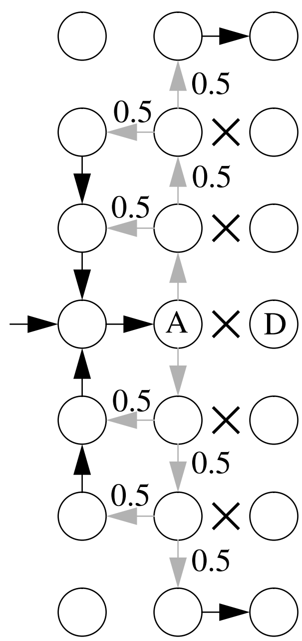


Figure 15: Extended linear obstacles. Dark arrow signifies route. Lighter arrow signifies deroute. Bypass of obstacle becomes more difficult but remains feasible as obstacle increases in size.

5.2.3.2 Global Reconfiguration

In order to preserve information on faults for the long term, each node must communicate fault data to a central node that acts as controller for the configuration. Since a single controller may be faulty, a set of redundant controllers can be used. Each controller constructs a map which charts the health of all components in the network. This information can be used for several purposes:

- Global information on faults is saved so that it can be taken into account when the system is reconfigured after a system reboot.
- Fault identification can be communicated to the system.
- The system administrator can be alerted about new faults.
- Dead or unreachable nodes can be reconfigured out of the system completely. Dead nodes should never be the destination of any messages.
- Messages destined for dead nodes can be removed.

With knowledge of system status with respect to faults, efficient operation of the network can be maintained despite the presence of many faulty components.

Certain combinations of faults present obstacles that the Chaotic routing algorithm cannot reliably avoid, even with the modifications described above. If the obstacle creates a concave faulty region of the network, as shown in Figure 16(a), then the routing algorithm will not be able to clear the obstacle and reliably deliver the packet except through the basic derouting mechanism with all its attendant inefficiencies. The problem lies in the fact that escaping such a region through intent requires a sequence of deroutes and the specific avoidance of certain locally profitable but globally unprofitable routes. To intentionally avoid a profitable route would require that a routing history be carried with the packet. This is contrary to the principles of the Chaos algorithm which uses only local information about the current state of the network. The mechanism for avoidance of linear obstacles requires no extended state but only knowledge of the direction in which the router was entered in order to function but that is not sufficient when confronted by concave faulty regions. Extension of the Chaos algorithm to incorporate a routing history was rejected because of the large increase in complexity required to incrementally extend the performance of the network.

The system software can provide some relief for this type of situation. The network map that is generated can be used to identify concave regions. Since there are no spares to

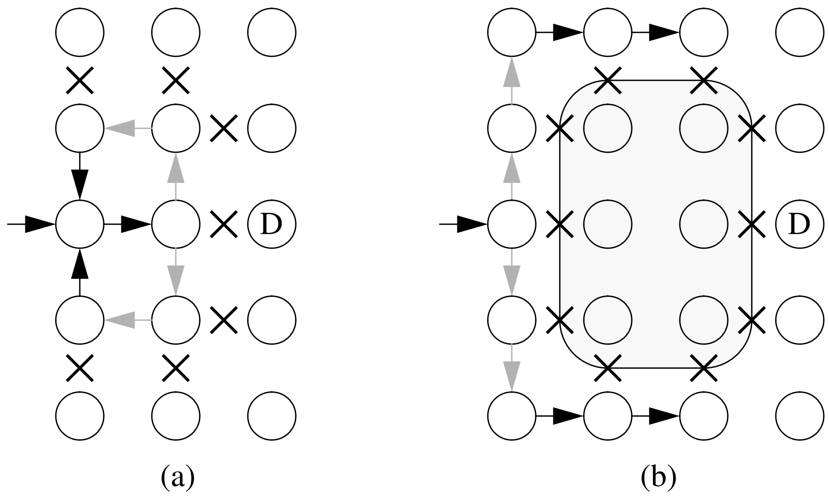


Figure 16: Livelock caused by concave region. Dark arrow signifies route. Lighter arrow signifies deroute. (a) Example in which concave region is not modified. (b) Example in which concave region has been blocked off (shaded region).

replace faulty components in the concave region, the only way to mitigate the problems caused by such a region is to close off the region such that the new boundaries of the obstacle are at least locally linear if not convex. This is shown in Figure 16(b). The central controller can send messages telling the appropriate nodes to mark certain links as dead. The nodes within the concave region end up isolated and unusable but the resulting configuration will be routable. This fix is costly in network resources so its usefulness will be on a spot basis, enabling the user to put off repair of the system when small concave regions form.

The problem of handling concave regions was addressed in a discussion of planar adaptive routing by Chien with a similar solution [Chien & Kim 92]. However, Chien defines a local algorithm that reconfigures the network to eliminate concave faulty regions. without network intervention. If a node has channels that are marked as faulty in each of two dimensions, then that node marks all of its channels as faulty. This algorithm is appealing because the use of a local algorithm is philosophically in tune with the Chaos routing algorithm and local distributed algorithms tend to be robust with respect to fault tolerance. However, this algorithm eliminates more components than required for the Chaos fault tolerant architecture to remain functional. Consider the situation in Figure 17(a). The local reconfiguration algorithm would close off nonfaulty nodes as shown in Figure 17(b). Figure 17(c) shows that the Chaos routing algorithm with the “no routeback” feature avoids the obstacle without any extra intervention into the system configuration. Since reconfiguration, local or global, requires that the system software have a capability to repartition the tasks undergoing computation which in turn requires that the network be mapped, this mapping and partitioning procedure can simply be extended to handle the global reconfiguration. The cost in performance is slight because reconfiguration should be a relatively rare event. Rarer still is the scenario in which concave faulty regions form.

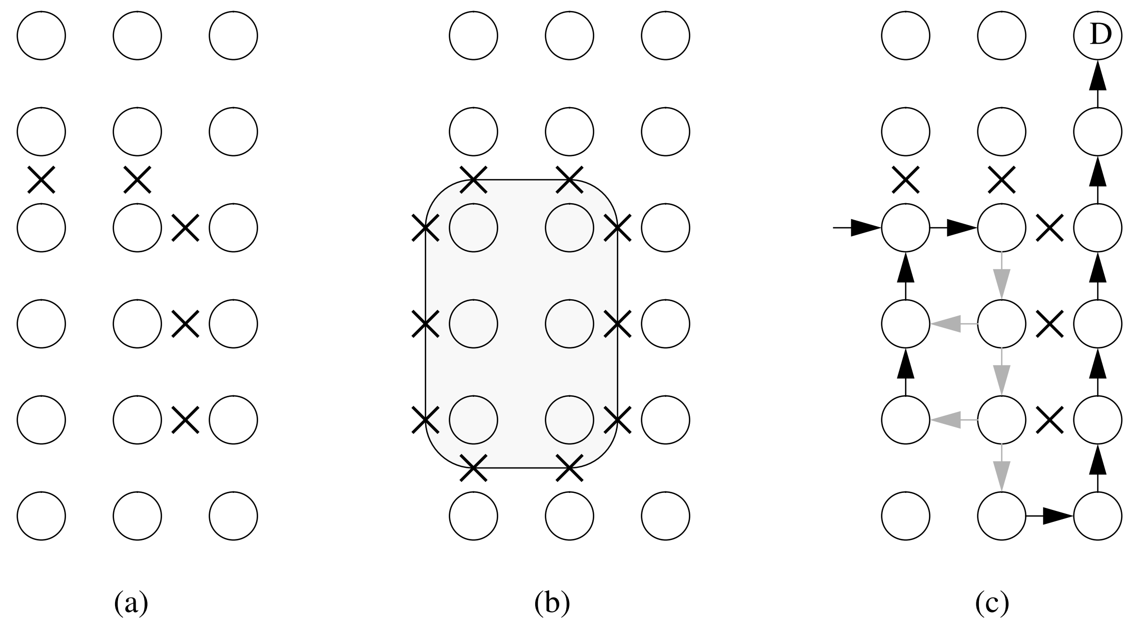
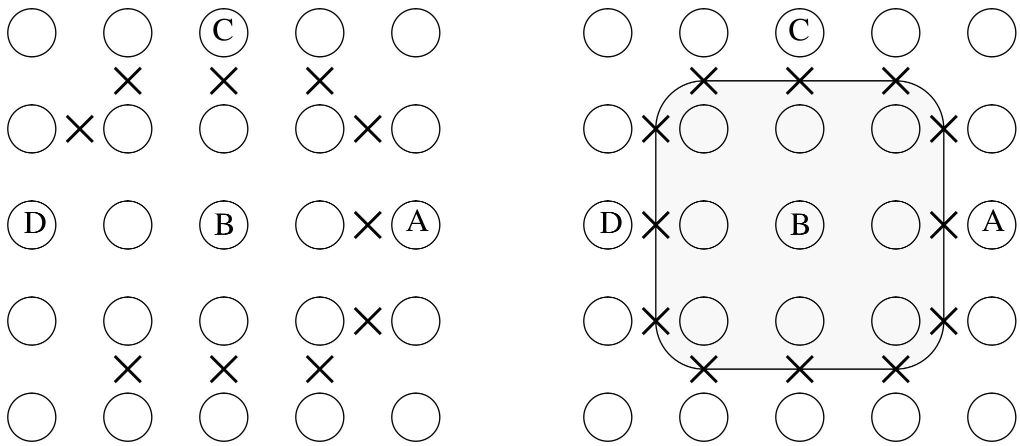


Figure 17: Planar adaptive routing global reconfiguration. (a) Original configuration. (b) After planar adaptive routing reconfiguration. (c) Chaotic routing bypasses the obstacle without any additional global reconfiguration.

The central node that maps the network may not be able to maintain communication with all connected nodes. The preliminary network map that it creates will have three states for components: known good, known bad, and unknown. Consider the situation in Figure 18(a). When node A attempts to map the network it may not receive any status information from nodes in the vicinity of node B. The solution to this problem is to divide and conquer, Node A can delegate mapping responsibility for subregions to other nodes that may be able to establish communication with nodes that node A cannot reach. In this scenario, node A delegates mapping responsibility to node C which is able to extend the known region. However, the status around node B remains unknown. Node C then delegates to node D which can communicate with node B and map the surrounding concave region. Node D can close off the concave region around node B which results in a configuration with all known boundaries shown in Figure 18(b). All nodes that remain connected in this configuration will be able to communicate. In the general case, each new delegation will not necessarily extend the known region, however, the delegation process can be performed progressively until all components have known status.



(a) (b)

Figure 18: Network mapping example. (a) Node A is unable to map the region around Node B. (b) After delegation of mapping authority, entire network is mapped and concave faulty region has been closed off (shaded region).

The ability to use system software to configure the system allows for the hot replacement of system components. If a board needs to be swapped, the network connections to that board can be disabled and the board removed without having to shut down the remainder of the system. The active nodes need to be informed which node is to be replaced so that all old communications can be completed and all new communications inhibited. Once the replacement board is installed, the replacement node is rebooted and communication to the rest of the system is reestablished.

5.3 Summary of Hardware Costs for the Fault Tolerant Architecture

The following is meant to be a rough order-of-magnitude analysis of the implementation costs of the fault tolerant Chaos architecture. Representative values used for the gate equivalent cost of structures such as registers and counters are meant to be generic in nature and are not associated with a specific choice of chip fabrication technology. The basic Chaos router as fabricated uses a mixture of standard cells and custom blocks which further confuses the issue. The basic router has about 75,000 transistors which translates into 20,000 gates as a baseline value. Estimates for the various circuit structures defined in the fault tolerant architecture are shown in Table 4.

The listed structures within the router amount to a total of 2572 gate equivalents. Adding 20% for glue logic makes the total about 3100. This adds 15.5% to the original gate count. The extra cost to provide full fault detection coverage through duplication of circuit blocks as done in the Vulcan design is 100%. The cost of reconfiguration and redundancy management is on top of that.

The cost to the Network Interface comes to about 4300 gates though as a communications coprocessor, the functions involved should probably be considered as part of the baseline design. The cost in wiring on the circuit board comes to one wire/link which adds 5% to the link hardware cost but less on an overall board wiring basis. On a packet basis, the architecture requires three added flits/packet. Two are associated with the checksum with one associated with required header fields. These flits are not necessarily extra as the basic router assumes a number of overhead flits in each packet, however, for the sake of a conservative analysis these flits can be considered a cost of the architecture. The cost in throughput then is three flits out of twenty or 15%. The cost in clock speed cannot accurately be estimated without carrying the design through a gate level implementation, however, the architecture was designed under the principle that the critical paths would not be significantly lengthened.

Table 4: Hardware implementation costs.

Mechanism	Discussion	Gate Cost
Router Chip:		
Error detection		
Data error checks		
Parity	15 XOR (4 gates) X 5 channels	300
Channel protocol checks		
Flit counter on input frames	5 bits (14 gates/bit) X 5 channels	350
Output frame timeout counter	8 bits X 5 channels	560
	8 bit register (10 gates/bit) to hold programmable value for output timeout	80
Channel controller checks	Incremental addition to controller state machine - (2 bits of state + 20 gates) X 5 channels	200
System drain		
Timeout counter	20 bit counter + 20 bit register	480
Logic to drop packets	Random logic	200
Reconfiguration		
Configuration register	9 bits (5 links + 4 routers)	90
Routing decision logic	Random logic	50
Fast derouting		
No routebacks		
EBN		
Input latch	1 bit X 5 channels	50
Controller	3 bits of state + random logic	100
Delay counter	2 X Network Delay = 2 X 5 X 16 for 1024 node system => 8 bit counter	112

Total		2572
Network Interface:		
Error detection		
Data error checks		
Checksum	Adder/Accumulator - 32 bits (16 gates/bit)	512
Lost packet check	16 bit counter X 16 channels + 16 bit register	3744

Total		4256
Circuit Board:	1 wire/link	

5.4 Reliability of the Fault Tolerant Architecture

The effectiveness of the fault detection mechanisms in fault tolerant architecture is shown by the results summarized in Table 5. The parity/checksum mechanism can only be defeated by multiple faults and error coverage is near 100%. For the contextual errors, 99% of packets are covered and a reliable end-to-end transmission protocol can provide coverage for the rest. For the channel protocol, coverage of overt errors is 100%. However, faulty channel behavior can mimic legal behavior and therefore some failures may escape detection. The sanity checks and other error detection mechanisms provide assurance that such undetected failures will be subtle, perhaps causing suboptimal routing performance, but will not result in corrupted data, loss of packets etc. Such failures that escape detection should be rare and should they occur, the resiliency of the network should still allow for continued network operation.

As discussed in Section 3, the translation of error coverage into fault coverage is necessary for the determination of system reliability. Fortunately, the fault classes that contain the router datapath and the data lines in the links are almost exclusively associated with the bit error class of errors. The translation of error coverage for corrupted data into fault coverage for the bit error class is essentially one-to-one.

The translation of error coverage into fault coverage for word and complex errors is more problematic. Faults in the routing decision logic fault class and the router control logic fault class may generate bit errors which are well covered. However, word and complex errors will also be generated and that coverage is harder to define. It is not that the coverage is low; there should be a good correlation between the high error coverage shown and the overall fault coverage. The high error coverage implies the coverage of word and complex errors will also be high. It is simply the mapping of those errors into the error detection scheme that is difficult to define.

Table 5: Network error syndromes.

Error	Detection Mechanism	Coverage
Packets with acceptable format but corrupted data.	Parity/Checksum	High
Packets with bad format:		
Truncated	Checksum	High
Stretched: ≤ 20 flits long	Checksum	High
> 20 flits long	Sanity check	Full
Context Violations:		
Misdelivered	Network Interface	Full
Misrouted	Network Interface	Late packets may be noted
Extra copies	Network Interface/ Reordering Buffer	High for multipacket messages
Dropped packet	Reordering Buffer	High for multipacket messages
Channel Protocol Violations		
Channel ownership transfer		
Never	Sanity Check	Full
Slow	Reordering Buffer	Packets may be noted as missing if bottle-neck forms
Transmission when nonowner is unable to receive packet	Protocol checker	Full
Impossible control sequence: Previously full input frame empties or previously empty output frame fills without transaction across channel	Protocol checker	Full
Loss of synch between channel controllers or between data and control.	Parity/Checksum/ Protocol checker	High

For this analysis a conservative range of coverage values is used to analyze reliability improvement. The range of 0.95 - 1.0 is used for fault classes that are associated almost completely with bit errors. The range of 0.5 to 0.9 is used for the other fault classes. The value 0.5 was chosen simply to show that even with very poor coverage, the overall results will still be favorable. The value 0.9 is on the low end of a reasonable but conservative estimate. Table 6 takes the definition of fault classes from Table 1 and adds the coverage information. The overall coverage for hard faults in the router chip range from 0.83 to 0.97. For hard faults in the links, the range is 0.86 to 0.98. The total coverage for hard faults is the average of these values and gives a range of 0.84 to 0.97. There is actually a weighting factor involved in this operation but it was ignored because the two ranges of coverage to be merged are very close and the relative weighting is difficult to determine. For soft faults the coverage ranged from 0.9 to 0.99.

Using the relation:

$$\lambda_{FT} = (1 - \text{coverage}) \times \lambda_{baseline}$$

the failure rate and the reliability of the fault tolerant network can be compared to the baseline network. This is shown in Table 7 for a 1024 node network. The improvement to reliability achieved by the fault tolerant architecture is on the order of two orders of magnitude for the conservative but reasonable estimate. An overall MTTF that is on the order of hundreds of days provides enough reliability such that the failure rate associated with the interconnection network is probably down in the noise relative to the overall multicomputer failure rate.

Table 6: Chaos network fault coverage.

Fault Class	Percentage	Coverage		Weighted Coverage	
		Low	Moderate	Low	Moderate
Router - hard faults					
datapath	75	0.95	1.0	0.71	0.75
routing decision logic	10	0.5	0.9	0.05	0.09
router control logic	15	0.5	0.9	0.07	0.13
Coverage - Router hard faults				0.83	0.97
Links					
data lines	80	0.95	1.0	0.76	0.80
control lines	20	0.5	0.9	0.10	0.18
Coverage - links				0.86	0.98
Coverage - all hard faults (router+links/2)				0.84	0.97
Router - soft faults					
datapath	90	0.95	1.0	0.85	0.90
routing decision logic	4	0.5	0.9	0.02	0.04
router control logic	6	0.5	0.9	0.03	0.05
Coverage - Router soft faults				0.90	0.99

Table 7: Reliability comparison - baseline vs. fault tolerant 1024 node network.

	Baseline Network		Coverage		Fault Tolerant Network				$MTTF_{FT}$: $MTTF_{base}$	
					λ		MTTF			
	λ	MTTF	Low	Mod erate	Low	Mod-erate	Low	Mod-erate	Low	Mod erate
Hard Faults	1.79×10^{-3}	557	0.84	0.97	0.286×10^{-3}	0.054×10^{-3}	3497	18520	6.3	33.2
Soft Faults	16.4×10^{-3}	61	0.90	0.99	1.64×10^{-3}	0.164×10^{-3}	610	6100	10	100
Overall	18.2×10^{-3}	55			1.93×10^{-3}	0.218×10^{-3}	518	4587	9.4	83.4

6 System Operation

It has been shown how software interacts with the network hardware in the fault tolerant architecture to provide a fault tolerant capability to the system. Most of the software procedures described fall into the category of local applications. It falls to the system software to tie the various mechanisms and procedures into an effective system.

The fault tolerant architecture provides a basis for effective fault detection and identification and efficient redundancy management. However, referring back to the goals of fault tolerant routing, the procedures and mechanisms described to this point do not by themselves provide for reliable message delivery. Packets in the network can become corrupted or lost. Information redundancy is necessary if the loss of a message is to be avoided. However, this design avoids schemes in which the hardware maintains copies of packets within the network. A reliable data link transmission protocol, in which the sender maintains a copy of the packet until it is acknowledged, can be used on a spot basis to aid in the detection of network failures. But this solution, which provides for reliable message delivery, is deliberately excluded from the general case because of the performance costs involved. A checkpoint/rollback solution turns out to have a nice implementation using the EBN, does not affect any critical paths and can be tuned to an appropriate frequency depending upon the actual system error rate observed.

The following section contains a high level overview of the system fault tolerance.

6.1 Putting It All Together

When the multicomputer is initialized, the latest system configuration needs to be restored so that all components previously marked as dead remain so marked. Old and identified faults must be isolated quickly to avoid having the equivalent of multiple faults applied to the system at once. Each node will maintain a configuration vector in local non-volatile memory. This vector will be applied to the router to restore the latest local configuration. If the router is unable to correctly apply the vector or the vector is in error, then a channel may activate in error. Such an activation will not be enough to restore a faulty channel because the neighboring node must also restore the channel in order for service to be restored. Otherwise, the error detection mechanisms on the channel will eventually detect a lack of response which will shut that channel down via an alternate means.

The central node or nodes that generate the network map will have final say over the initial configuration. A new map is generated which is compared to the old map. The two maps are merged with only those components alive in both maps left alive in the composite map.

Appropriate messages are sent to individual nodes telling them to mark certain links as dead. Finally, configuration and partition control information is disseminated across the network. An enhanced EBN could be used to efficiently perform this task as described in Section 4.4.1.

Restoration of a previously faulty but repaired network component to active status requires intervention by the system software. If the repaired component is simply a link that had been marked as inoperative by the nodes at both ends, then messages directing those nodes to relabel the link as in service are all that is required. If the repaired component is an entire node that had previously been isolated, then all neighboring nodes of the repaired node must be informed that the connections to the repaired node should be restored to service. The repaired node must initialize in a state that allows communication with its neighbors so that it can receive current system configuration and partitioning information. In particular, if the repair is done to a hot system, then the system software must dynamically repartition the system when the repaired component comes on line.

Once the system commences computational activity, *checkpointing* is periodically undertaken. In combination with a *rollback* procedure, checkpointing provides the information redundancy that ultimately provides for reliable message delivery. The dynamic reconfiguration capability of the network requires a dynamic repartitioning capability which in turn requires that a “good” computational state be available. Hence, checkpointing becomes imperative to avoid the situation in which the only known “good” state is the initial state.

Checkpointing provides a snapshot of the system state such that computation can be restarted from an intermediate “good” state if the computation is corrupted by an error. Instead of rolling back the computation all the way to the start, the checkpoint enables the preservation of the uncorrupted results of earlier phases of the computation so that they do not have to be recalculated. The amount of state that must be preserved depends upon the specific process in progress. Checkpointing can be facilitated through use of the EBN. The EBN barrier function can be used to establish the condition for initiation of the checkpoint. The EBN barrier can be used again to mark completion of the checkpoint.

The rollback is an integral part of the procedure for handling transient errors. The diagnostics and reconfiguration procedures are required for the identification of hard failures and for redundancy management, however, transient error management requires only detection and recovery. For those errors that cannot be cleared by a lightweight response such as a request for retransmission, the rollback provides a heavyweight fallback recovery method.

There are different amounts of activity required to perform a rollback depending upon the circumstances. If reconfiguration has not disconnected pieces of the network, then a simple rollback in which each node restores its own checkpointed state can be performed. Because the architecture allows for nodes to become isolated from the rest of the system, there must be some node-to-node redundancy of checkpoint data so that the computational state can be recovered even if the physical network configuration has changed between the time at which the checkpoint was performed and the rollback time.This situation requires a more complex rollback procedure. The state from the isolated node must be doled out to the remaining nodes followed by a dynamic repartitioning of the computation across the multicomputer system.

Error detection is performed almost without intervention by the system software. The router state, as defined in Section 4.5, is monitored by the processing nodes so that they may track the fault management procedures and intervene as required. Intervention by the system software will only occur at the tail end of the process. From a high-level viewpoint, the system drain that results from an error-detected eureka is simply low-level network flow control which temporarily slows the system but does not have a direct effect on overall system function. When the drain is complete, any of the processing nodes can drive the system into the diagnostic procedures which are run as local applications. Only in compiling the network map at the end of the diagnostics and performing any global reconfiguration does the system software enter the process. The decision to rollback the computation is contingent on the results of the diagnostics and the type of errors detected. If a message is lost or corrupted and the destination node is unable to request a retransmission, then the result will be a rollback. A reconfiguration of the network that results in the loss of a node or nodes will also automatically result in a rollback. A reconfiguration that shuts down links but does not disconnect the network makes a more difficult decision necessary. A failure detected by means of the diagnostics does not necessarily imply that data has been lost or that the computation state is corrupted. However, it is an indirect indication of such a problem and the safest choice in this situation is to rollback. Reconfiguration will be an infrequent event so the performance cost of the rollback in this situation will be slight. Errors in which data is not lost or can be recovered outside of the rollback and in which diagnostics do not uncover any new failures do not require a rollback.

The local configuration vector needs to be kept up-to-date. Any change to the vector, either generated by the diagnostics or directed by the system software will require an update of the vector in non-volatile memory. This is also the case for the network configuration map.

7 Conclusions

The following summarizes the contents of this thesis. Also included is a discussion of some directions for future research.

7.1 Summary

A design that greatly enhances the fault tolerance of a Chaotic routing while avoiding large added costs has been presented. As an adaptive non-minimal packet routing algorithm, Chaotic routing provides a natural means by which obstacles can be avoided. Faulty components mimic congestion and when congestion is detected, the Chaos router responds by derouting packets in a non-minimal direction to avoid the congested region. However, this places the router in its most inefficient operational regime. In addition, other types of errors resulting from faults may be manifested other ways, with symptoms such as dropped or corrupted packets. Hence, the basic router cannot provide for robust fault tolerant routing.

The design space of interest for this study was defined. The twin constraints of cost and performance drive the design for fault tolerance into the space of maximum added reliability for the least cost rather than extremely high reliability combined with high added costs. A single sequential fault model for the network was defined. Estimates of the failure rate of the basic Chaotic routing network were made. For hard faults, a MTTF for the network of 557 hours was estimated. For soft faults, a MTTF of 61 hours was estimated.

The fault tolerant architecture selected uses procedures and algorithms for fault management that are distributed and local but synchronized across the network. An efficient broadcast mechanism for network synchronization is necessary. The Express Broadcast Network was presented as a dedicated broadcast network that provides highly reliable fault tolerant service with low cost, low latency and performance independent of the data network load. The eureka and barrier functions which form the basic network primitives provide for efficient network control of many possible applications. The basic EBN can be implemented as a single wire per network link. Extensions to the EBN provide for synchronous one-to-all broadcasts. In the fault tolerant Chaotic routing architecture, the basic EBN implementation provides for control and synchronization of the network fault management procedures.

The basic fault management procedure in the Chaos routing network performs fault detection every cycle. A detected error forces the system into a system drain procedure in which packet injection is inhibited and all packets delivered or removed from the network. Diagnostics are performed to determine which links or processing nodes may be faulty. System reconfiguration, in which newly diagnosed faulty components are disabled, follows. A roll-back may be necessary to restore a known good state to the system. The system then returns to normal operation.

Fault detection within the Chaotic routing network depends upon the use of a limited number of detection mechanisms at carefully selected spots within the router. A comprehensive scheme in which all circuitry is duplicated and outputs compared was avoided because of the cost. The selected mechanisms fall into two categories: 1) those that detect explicit errors that escape from a node, specifically form and content errors in packets, and 2) context errors in which packets may be late, missing, duplicated or misdelivered. The first category includes parity/checksum checks which guard against corrupted data and channel protocol checks which check such things as the length of packets and the protocol followed by the channel control signals.

The second category is associated with the reordering buffers within the network interface between the router and the processing node. Missing packets that are part of a multipacket message are detected when a hole appears in the contents of the reordering buffer. This mechanism does not detect missing single packet messages. However, that problem was analyzed and single packet messages were found to comprise less than 1% of network traffic under a typical work load. This means that the spot use of a reliable end-to-end transmission protocol utilizing buffering and message receipt acknowledgments would have only a small effect on network performance.

The system drain ensures an upper limit on packet delivery time within the network and sets the network into a known, empty state before the onset of diagnostics. Packet injection is inhibited by the drain and packets already in the network are allowed to proceed. As the network clears, obstacles caused by congestion will disappear and delayed packets will eventually be able to proceed. Faults may prevent the delivery of some packets within the network so, after a timeout period, packets that remain in the network are dropped.

Network diagnostics are designed to identify as many faults as possible using a quick distributed algorithm. The router is exercised in a controlled manner which exercises most of the router data path and all of the network links. Each node injects packets destined for each of its neighbors. When a node receives such a packet, it will in turn respond with an acknowledgment packet back to the original sender. Failure of an acknowledgment to return or the return of a corrupted packet is evidence of a fault.

Each node compiles the results of the local diagnostics and marks, in the router's internal configuration register, whether or not its links and neighboring nodes remain operational. This information is then used as a mask for all subsequent routing decisions made by the router. The system software also compiles the local configuration information from each node into a overall network configuration map. At that level, large obstacles that present routing problems can be identified and global network reconfiguration performed in order to remove routing bottlenecks.

The local map maintained by the node is used in several ways. Packets will not be routed in directions with unavailable resources even if that direction is part of a minimal path in the fault-free network. If no minimal path remains operational, then fast derouting will route a packet out an alternate non-minimal path without requiring entry into the router multiqueue. This eliminates the extra latency associated with buffering in the multiqueue.

If at the end of the diagnostics it is determined that a node has been lost to the system or if a packet has been lost in an unrecoverable manner, a rollback will be performed to return the system to a known good state. Otherwise the network will simply return to the normal operating mode and resume packet injection and routing activity.

The cost to implement the fault tolerant architecture has been estimated to be an additional 15% in router gate count. The cost in added wiring amounts to one additional wire or 5% per link. The cost in network clock rate is small as the critical paths through the router are largely unaffected by the modifications. The cost in throughput is 15% as an outside estimate because of the number of flits in a packet dedicated to fault tolerance. However, the basic Chaos router assumes a small number of overhead flits in each packet so this cost may not actually represent a real increase.

A translation of error coverage into fault coverage is necessary for the calculation of network reliability. While the translation is difficult to define precisely, high error coverage should correlate with high fault coverage. Conservative estimates of fault coverage show that the network MTTF improves from one to two orders of magnitude.

For hard faults, estimated MTTF improved from 557 hours to 18520 hours; a factor of 33.2. For soft faults, MTTF improved from 61 hours to 6100 hours; a factor of 100. Overall MTTF for hard and soft faults together went from 55 hours to 4587 hours; a factor of 83.4.

In summary, a design has been presented that provides for significant improvements in network reliability, can be implemented with only incremental changes to the basic Chaos architecture, has benign effects on fault-free system performance and only requires a fractional increase in the network cost associated with hardware resources.

7.2 Future Work

The design of the fault tolerant Chaotic routing architecture provides a framework for a reliable network implementation. However, there remain issues worth investigating, many associated with the details of carrying the design to a lower level of abstraction while some result from the limitations imposed on the design by the need to avoid costly changes to the basic design. Among the issues:

•**Improved diagnostics.** The diagnostics described center around data errors and faulty links. Coverage of router failures is more problematic. The test patterns used for fault diagnosis do not provide full coverage of the router data path and leave much of the router control untested.

•**Error control coding**. While this study makes a proposal for the use of a checksum to protect packet data, the choice will really depend upon the application and implementation. Codes can be tailored for most any need and an examination of error rates and failure modes that corrupt data will provide better insight into the amount of protection required from an error control code.

•**EBN fault tolerance**. While the EBN is inherently reliable due to the redundancy of the network, it has failure modes that need to be avoided. The physical implementation will require some sort of protection in order to locally contain failures. For example, the addition of a mechanism such as a heartbeat would allow for low-latency detection of a faulty EBN link.

•**Enhanced router fault detection**. There are additional means by which internal router faults can be detected without requiring brute force duplication. For example, the addition of parity to the packet as it passes through the router datapath could provide for an internal end-to-end check and low latency fault detection. Controllers can be built with extra state bits that expand the state space and provide the equivalent of parity. An examination of the trade-offs relating design effort and router area versus better fault coverage and improved reliability could be worthwhile.

•**EBN studies**. While the basic EBN network was simulated and debugged as part of the fault tolerant architecture design, simulation studies of the enhanced version have yet to be performed. Undoubtedly there remain many details to wring out.

•**Effects of system scaling**. The utility of global fault management procedures depends upon the the network failure rate which is a function of the individual component failure rates along with the size of the system. Global procedures remain practical as long as such procedures use only a tiny fraction of system computational resources. If this condition cannot be met, then an alternate scheme which limits the scope of such procedures will need to be developed.

•**System implementation**. Ultimately, reliability is a historical measure. Unless a system is actually built, there is no guarantee that all failure modes have been considered or that the trade-offs performed in a paper design have a legitimate basis. Somewhere in architectural and conceptual studies there needs to be an empirical basis for the decisions and choices that are made.

Bibliography

[Allen et al. 94] J. Allen, P. Gaughan, D. Schimmel, S. Yalamanchili. Ariadne - An Adaptive Router for Fault-tolerant Multicomputers. In *Proceedings of the International Symposium on Computer Architecture*, pages 278-288, 1994.

[Bhagwat et al. 94] P. Bhagwat, P. Mishra, S. Tripathi. Effect of Topology on Performance of Reliable Multicast Communication. In *Proceedings of the Conference on Computer Communications*, pages 602-609, 1994.

[Blough & Pelc 93] D. Blough, A. Pelc. Diagnosis and Repair in Multiprocessor Systems. In *IEEE Transactions on Computers*, pages 205-217, February 1993.

[Bolding 93] Kevin Bolding. Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router. PhD dissertation, University of Washington, Seattle, Wa., July 1993.

[Chen & Hsiao 84] C.L. Chen, M. Y. Hsiao. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. In *IBM Journal of Research and Development*, pages 124-134, March 1984.

[Chien & Kim 92] A.A. Chien, J. H. Kim. Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 268-277, 1992.

[Cray Research 93] Cray Research. *Cray T3D System Architecture Overview Manual*, 1993.

[Cypher et al. 93] R. Cypher, A. Ho, S. Konstantinidou, P. Messina. Architectural Requirements of Parallel Scientific Applications with Explicit Communication. In *Proceedings of the International Symposium on Computer Architecture*, pages 2-13, 1993.

[Dally et al. 94] W. Dally, L. Dennison, D. Harris, K. Kan, T. Xanthopoulos. The Reliable Router: A Reliable and High-Performance Communication Substrate for Parallel Computers. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 241-255, 1994.

[Davis 92] A. Davis. Mayfly: A General-Purpose, Scalable, Parallel Processing Architecture. In *Lisp and Symbolic Computation*, pages 7-47, May 1992.

[Davis et al. 94] A. Davis, R. Hodgson, I. Robinson, L. Cherkasova, V. Kotov, T. Rokicki.

R2: A Damped Adaptive Router Design. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 295-309, 1994.

[Dutton 89] T. Dutton. Connector Failure - A Terminal Problem? In *IEE Colloquium on "Connectors on Vehicles"*, pages 3/1-6, 1989.

[Elder et al. 88] J.Elder, J. Osborn, W. Kolasinski, R. Koga. A Method for Characterizing a Microprocessor's Vulnerability to SEU. In *IEEE Transactions on Nuclear Science*, pages 1678-1681, December 1988.

[Garcia-Luna-Aceves 88] J. Garcia-Luna-Aceves. Reliable Broadcast of Routing Information Using Diffusing Computations. In *Proceedings of the Global Telecommunications Conference*, pages 615-621, 1992.

[Hopkins et al. 78] A. Hopkins, T. Smith, J. Lala. FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. In *Proceedings of the IEEE*, pages 1221-1239, October 1978.

[Ishihata et al. 91] H. Ishihata et al. Third generation message passing computer AP1000. In *Proceedings of the International Symposium on Supercomputing*, 1991.

[Iyer & Hsueh 90] R. Iyer and M. Hsueh. Analysis of Field Data on Computer Failures. In *Journal of Computer Science & Technology*, pages 99-108, Vol. 5 No. 2 1990.

[Kim et al. 94] J. Kim, Z. Liu and A. Chien. Compressionless Routing: A Framework for Adaptive and Fault-tolerant Routing. In *Proceedings of the International Symposium on Computer Architecture*, pages 289-300, 1994.

[Konstantinidou 91] S. Konstantinidou. Deterministic and Chaotic Adaptive Routing in Multicomputers. PhD dissertation, University of Washington, Seattle, Wa., May 1991.

[Lee & Shin 94] S. Lee and K. Shin. Interleaved All-to-All Reliable Broadcast on Meshes and Hypercubes. In *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5 No. 5 pages 449-458, May 1994.

[Lamport et al. 82] L Lamport, R Shostak, M Pease. The Byzantine Generals Problem. In *ACM Transactions on Programming Languages and Systems*, pages 382-401, July 1982.

[Leiserson et al. 92] C. Leiserson et al. The network architecture of the connection machine CM-5, In *Proceedings of Symposium on Parallel Algorithms and Architectures*, pages 272-285, 1992.

[McKenzie et al. 94] N. McKenzie, K. Bolding, C. Ebeling, L.Snyder. CRANIUM: An Interface for Message Passing on Adaptive Packet Routing Networks. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 266-280, 1994.

[MH217 91] MIL-HDBK-217F. Military Handbook, Reliability Prediction of Electronic Equipment. 1991.

[Olson & Shin 88] A. Olson, K. Shin. Fault-Tolerant Routing in Mesh Architectures. In *IEEE Transactions on Parallel and Distributed Systems*, pages 1225-1232, November 1994.

[Ramanathan & Shin 88] P. Ramanathan, K. Shin. Reliable Broadcast in Hypercube Multicomputers. In *IEEE Transactions on Computers*, pages 1654-1657, December 1988.

[Siewiorek et al. 78] D. Siewiorek, V. Kini, H.Mashburn, S. McConnel, M. Tsao. A Case Study of C.mmp, Cm*, and C.vmp: Part 1 - Experiences with Fault Tolerance in Multiprocessor Systems. In *Proceedings of the IEEE*, pages 1178-1199, October 1978.

[Siewiorek & Swarz 82] D. Siewiorek, R. Swarz. *The Theory and Practice of Reliable System Design.* Digital Press, 1982.

[Somani & Agarwal 92] A. Somani, V. Agarwal. Distributed Diagnosis Algorithms for Regular Interconnected Structures. In *IEEE Transactions on Computers*, pages 900-906, July 1992.

[Stunkel et al. 94] C. Stunkel, D. Shea, B. Abali, M. Denneau, P. Hochschild, D. Joseph, B. Nathanson, M. Tsao, P. Varker. Architecture and Implementation of Vulcan. In *Proceedings of International Parallel Processing Symposium,* pages 268-274, 1994.

[Tang & Chien 69] D. Tang, R. Chien. Coding for Error Control. In *IBM Systems Journal*, pages 48-86, Vol. 8 No. 1,1969.

[Wang & Schwartz 93] C. Wang, M. Schwartz. Identification of Faulty Links in Dynamic-Routed Networks. In *IEEE Journal on Selected Areas in Communications*, pages 1449-1460, November 1994.

[Wensley et al. 78] J. Wensley, L. Lamport, J. Goldberg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostak, C. Weinstock. SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. In *Proceedings of the IEEE*, pages 1240-1255, October 1978.