Simulative Analysis of Fault-Tolerant Distributed Switching Fabrics for SCI

Mushtaq Sarwar
HCS Research Lab[1]
Dept. of Electrical Engineering
FAMU-FSU College of Engineering
Tallahassee, FL
sarwar@hcs.fsu.edu

Alan D. George
HCS Research Lab
Dept. of Electrical and Computer Engineering
University of Florida
Gainesville, FL
george@hcs.ufl.edu

Abstract

Several common topologies such as k-ary n-cubes possess inherent link redundancy and can be directly applicable to systems demanding fault-tolerant networks. In this paper we present the latest results in an on-going research project to model, simulate, and evaluate SCI-based distributed switching fabrics for fault-tolerant, mission-critical applications. The SCI models developed for this research incorporate extensions to the existing SCI fault-detection protocols in order to locate, isolate, and recover from network failures. We present several fault-free, and fault-injection simulation performance results to demonstrate the effectiveness of these protocols in recovering from particular network failures. We also present reliability simulation results for 1D and 2D k-ary n-cube topologies used in the performance evaluations.

1.0 Introduction

Mission-critical, critical-computation, and other demanding applications can benefit from low-cost, high-bandwidth interconnects such as SCI. These applications also demand fault tolerance to maintain connectivity of all data processing nodes. In response to this need, we are currently developing and refining a fault-tolerant SCI modeling environment to design and evaluate both the performance and reliability of SCI interconnects.

To realize such a modeling environment required the development of three main components. The first is a flexible SCI model that is used as a basis for the fault-tolerant models and which is used to simulate fault-free SCI topologies. The models developed include SCI interfaces and agent-based switches that permit multiple I/O ports per node and support up to 64 outstanding transactions as defined by the standard. The models allow the user to configure any SCI topology and simulate the topology within minutes. Easy plug-and-simulate capability was the primary concern during development, hence routing tables are generated automatically based on a connectivity matrix initialized at system startup. Using routing tables based on a connectivity matrix has three main benefits. First, it provides topology independence requiring no specific routing algorithms to be introduced if the topology is altered. Second, it permits the embedding of virtual topologies within an underlying physical topology simply by altering the connectivity matrix. And finally, and most importantly, faulty rings can be eliminated from the routing tables by altering the connectivity matrix and rerunning the generating algorithm for the routing tables at each node.

The second major component required to develop the simulation environment involves the addition of fault-detection, location, and recovery protocols for general distributed switching fabrics for SCI. The inclusion of these protocols dictates the incorporation of additional fault-recovery modules within the switch model. Fault detection is performed using many of the existing fault-tolerance protocols described in the SCI standard, however recovery from these faults required the definition of new commands to diagnose and instruct nodes on how to recover from particular faults.

Once a fault-tolerant SCI topology has been constructed that satisfies the performance requirements of the user, it is important to determine the reliability of such a system prior to implementation. Hence the third and final component in the simulation environment is a technique to determine the reliability of the modeled fault-tolerant topology. The reliability is evaluated using UltraSAN, a Stochastic Activity Network (SAN) based package developed by Sanders et al. at the University of Illinois at Urbana-Champaign. A SAN-based approach was chosen due to the large number of interdependencies created when link failures occur. As the complexity of the topologies increase, so do the interdependencies between failed rings. Consequently, the state space of the Markov models used to measure the reliability grows exponentially with even the slightest increase in topology size.

[1] The High-performance Computing and Simulation (HCS) Research Lab is a joint research facility affiliated with the Univ. of Florida, Florida A&M Univ., and Florida State Univ., and serves as the *NSA Center of Excellence in High-Performance Networking and Computing*. See http://www.hcs.ufl.edu

The topologies currently used in the simulation environment consist of 1D and 2D *k*-ary *n*-cubes, but the environment is capable of modeling any ring-based SCI topology. In this paper we describe the models created and exploited to date, and present performance analyses, fault-injection simulation results, and reliability analyses for several *k*-ary 1-cube and *k*-ary 2-cube topologies.

The remainder of this paper is organized as follows. Section 2 describes the SCI switch model, routing algorithm, and fault-free simulation results. Section 3 discusses the fault-tolerance additions to the model and the results of several fault-injection simulations. Section 4 presents the UltraSAN reliability analysis results, and finally the conclusion and future directions of this research are discussed in Section 5.

2.0 SCI distributed switching simulations

This section describes the SCI switch model and its associated routing algorithm. The simplest of all SCI switches consists of two back-to-back SCI interfaces where the input queues of one feed into the output queues of the other and vice versa. This configuration is limited in that it only allows the bridging of two SCI ringlets. In order to interface more that two rings at a single point, a switch must provide additional SCI interfaces, one per attached ringlet.

Wu and Bogaerts have studied several SCI switch models consisting of internal rings, buses, and crossbars for use in multistage SCI networks [2-4]. Their results show that although high throughputs can be achieved using an internal bus, even better performance is attainable by using a crossbar-based switch. The two crossbar-based switches studied by Wu and Bogaerts are the CrossSwitch and Switchlink. The CrossSwitch provides a single output queue for each output port, regardless of the number of input ports. This structure requires input ports to block in cases where there is contention on an output queue. In contrast, the SwitchLink [5] provides one output queue for each input port at every output port. This multi-queue approach provides a private path from each input queue to one and only one output queue, hence eliminating contention for a single output queue and minimizing head-of-line (HOL) blocking.

The switch model developed for this research is based on the multi-queue concept used in the SwitchLink. Figure 1 illustrates the switch model in a 4-port configuration. By incorporating a crossbar and multiple SCI interfaces into each node, several rings can be connected at a single point. In addition, if the switch's ability is extended to service a processing unit, the resulting switch/node model can be used to construct and simulate any standard ring based topology that utilizes distributed switching. Examples of such topologies include dual-ring topologies, the *k*-ary *n*-cube family of topologies, and Manhattan street networks. The basic premise of these topologies is to use each node not only as a processing unit but also as a switching point for packets.

The final addition needed to the model is a routing algorithm. The following subsection describes the algorithm developed to generate the routing tables.

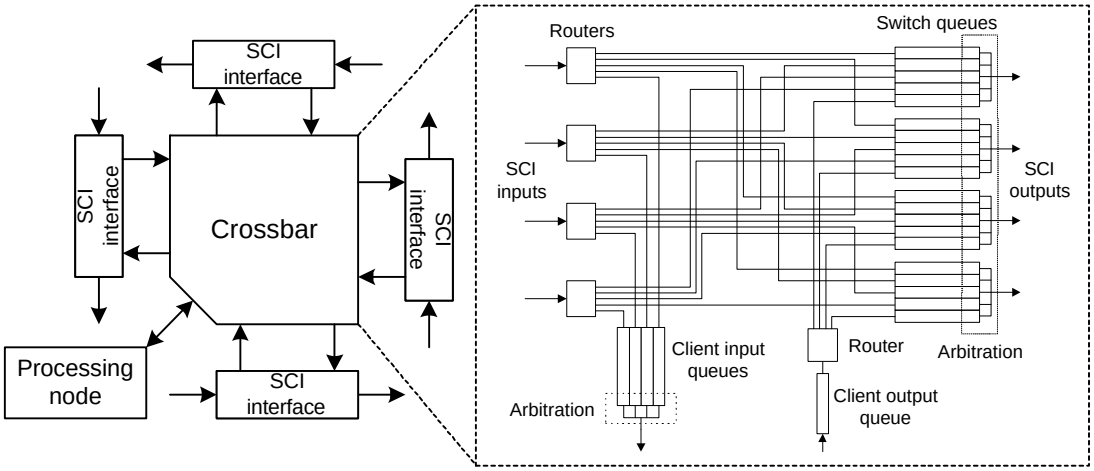


Figure 1. Switch model in a 4-port configuration

2.1 Routing

One of the key elements in the model is routing. A number of routing algorithms for *k*-ary *n*-cubes including mesh topologies have been presented in the literature. Examples include the work by Hou et al. [6], Gravano et al. [7], Chein et al. [8], and Fang et al. [9]. One limitation of these algorithms is that they are function-based and hence specific to particular topologies. In order to avoid the fixed-topology restriction, table-lookup routing is used in the models. The use of table-lookup routing permits the user to configure any topology using the model building blocks without worrying about routing issues.

The algorithm described in this section generates a set of routing tables for any point-to-point topology with unidirectional or bi-directional links. Naturally it also extends to ring based point-to-point topologies. The routing tables generated by the algorithm collectively define the shortest path from any source node to any destination node, if one exists. Unlike the Dijkstra and Ford shortest-distance algorithms [10], the path length is measured in terms of traversed links, each of equal weight. This simplification makes it appropriate for use in multiprocessor systems and workstation clusters using equal length interprocessor links. The routing tables indicate which of the switch output links provides the initial step in the shortest path from the source node to the destination node. By repeating the lookup at all intermediate switch nodes in the path, the packets arrive at the destination in the fewest possible number of hops.

As an example, consider the six-node network illustrated in Figure 2, consisting of both unidirectional and bi-directional links. The numbers immediately outside each node represent the output port numbers for that node. The algorithm assumes that the smaller node numbers are connected to the smaller port numbers. For example, in the case of node 3, output port 1 is connected to node 2 and output port 2 is connected to node 4. Since the port numbers are mapped in software within the switch, this assumption does not affect the network or routing tables in any way.

To generate the routing tables, the topology of a network must first be described using a connectivity matrix *C*. The connectivity matrix for an *N*-node network is an *N* x *N* matrix. This matrix describes the topology of the network. If a link exists between two nodes, a 1 is placed in the element position at which the source and destination node intersect in the matrix. In the case of a bi-directional link between a source node *s* and destination node *d*, $C(s,d) = C(d,s) = 1$. If a unidirectional link exists from *s* to *d*, then $C(s,d) = 1$ and $C(d,s) = 0$.

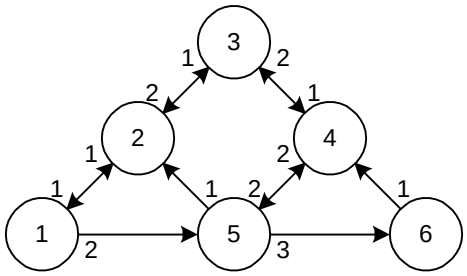


Figure 2. Sample topology

The nodes in the example network are numbered 1 through 6. The connectivity matrix *C* for the network in Figure 2 can therefore be written as:

$$C = \begin{array}{c} \\ \text{Source nodes} \end{array} \begin{array}{c} \text{Destination nodes} \\ \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 1 & 0 \\ 5 & 0 & 1 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 1 & 0 & 0 \end{array} \end{array}$$

Once the connectivity matrix is obtained, either statically by the system designer or dynamically by the switches themselves, it is broadcast to all nodes. The following algorithm is then executed at each node *n* to generate the routing table for node *n*.

Step 1

At each node *n*, a 1 x *N* link vector L_n is extracted from the connectivity matrix *C* such that $L_n(j)=C(n,j)$ for $j=1, 2, \ldots, N$. This vector has a 1 in the position of the nodes to which node *n* has a direct link. For example, in the topology shown in Figure 6, node 4 would extract $L_4 = [0\ 0\ 1\ 0\ 1\ 0]$.

Step 2

Next the link vector L_n is used to construct two *N*x*N* search matrices S_{n1} and S_{n2} at each node *n*, where each element *j* of row *i* in S_{n1} and S_{n2} represents the outgoing port number used to route to node *j* in *i* hops. A zero entry indicates that no path to node *j* in *i* hops exists. Each search matrix defines the first hop of one path to each destination. Two search matrices

are used in order to find two equidistant paths from the source to the destination nodes, if they exist. Multiple paths are provided to improve load balancing. The algorithm can be easily expanded to construct more than two search matrices and in turn locate more that two equidistant paths to each destination if they exist. The search matrices S_{n1} and S_{n2} at each node n are constructed by following two steps:

a. The first step is to fill in the first row of each search matrix. Each subsequent row r is then filled using row $r-1$ of S_{n1} and the connectivity matrix. For each occurrence of a 1 in the row vector L_n, unique identifiers are placed in the first row of the search matrices in the columns indicated by the positions of the 1s in L_n. In this case, the port numbers of the nodes are used as identifiers. The algorithm to complete the first row of the search matrices is as follows:

```
for j=1 to N
    if Ln(j)=1
       Sn1(1,j) = Sn2(1,j) = node's port number
    endif
endfor
```

The remainder of the entries in the first row of the search matrices are set to 0 indicating that no direct path to those nodes exist from node n. Continuing with the example of node 4, the first row of S_{41} and S_{42} would become [0 0 1 0 2 0], where 1 is the port number connected to node 3 and 2 is the port number connected to node 5.

b. Once the first rows of the search matrices have been filled, each is used in conjunction with the connectivity matrix C to fill the remainder of the matrices using the following algorithm.

```
for i=1 to N-1
    for j=1 to N
        if Sn1(i,j)¹0
            L(x)=C(j,x) for x=1 to N
            for y=1 to N
                if L(y)¹0 and Sn1(i+1,y)¹0
                    Sn2(i+1,y)=Sn2(i,j)
                endif
                if L(y)¹0 and Sn1(i+1,y)=0
                    Sn1(i+1,y)=Sn1(i,j)
                endif
            endfor
        endif
    endfor
endfor
```

Step 3

The routing table (*Route*) can then be constructed from the search matrices. The resulting routing table represents a breath-first search tree from the source node to every reachable node.

```
for j=1 to N (j¹ my node number)
    for i=1 to N
        if [Sn1(i,j)¹0 and (Sn1(x,j)=0 for all x=1 to i-1)]
            Route(j,2)=Sn1(i,j)
            Route(j,3)=Sn2(i,j)
            Route(j,4)=i
        endif
    endfor
endfor
```

The algorithm of step 3 assumes that the first column of the routing table is already initialized with the destination node numbers. A value of 0 in element (i,j) of S_{41} and S_{42} indicates that no path to node j in i hops exists. Since only equidistant routes are considered, the routing table for node 4 can then be constructed using S_{41} and S_{42} as:

Destination Node	Route 1	Route 2	Number of links traversed
1	1	2	3
2	1	2	2
3	1	0	1
5	2	0	1
6	2	0	2

If an entry in the route 2 column is zero, then it indicates that a second equidistant path to the destination node does not exist. For example, in the entry for destination node 5, a route 1 column entry of 2 indicates a path from node 4 to destination node 5 via port 2. The 0 entry in column route 2 indicates that a second equidistant path to the destination does not exist.

2.2 Fault-free simulation results

This section presents simulation results both to verify the correctness of the model and to evaluate the performance of SCI in various topologies. The models were constructed using the Block Oriented Network Simulator (BONeS), a commercial CAD tool from Cadence Design Systems [11-12]. BONeS is an event-driven simulation engine primarily used for modeling and analysis of computer systems and communications networks. Models are created using core library blocks and custom-built blocks. Each block has an equivalent C++ implementation that when executed, simulates the function of the block. The BONeS model created for this research is an

Table 1. Model parameter assignments

Parameter	Value
Link Speed	1 GB/sec (as defined by the SCI standard)
Switch Queue Lengths	5 packets deep
SCI Node Queue Lengths	5 packets deep
Routing Decision Time	10 ns
Packet Switching Time	2 ns per 2-byte symbol
Clock Frequency	500 MHz
Packet Destinations	Uniformly distributed random destination
Transaction Type	64-byte Move (64-byte payload, 16-byte overhead)
Mean Time Between Packets	64 / (Total Offered Load / *N*)

extremely high-fidelity performance model. The SCI protocol is modeled down to the bit level and functionally implemented within the model. Unlike coarse-grained models that assume a fixed packet size and hence fixed switching, queuing, and transmission times, the BONeS model relies on the packet size to measure time.

The switch/node models constructed in BONeS consist of two-port, three-port, and four-port variants. The functions of the switch include servicing the processing node as well as providing switching capabilities to the entire network. Table 1 shows the value of several parameters common to all simulated topologies.

Most of the parameters in Table 1 are self explanatory, but a few are worth noting. The routing decision time is the penalty imposed by a multi-port node when a packet must be switched off one ring and onto another. The packet switching time is the amount of time it takes to switch one 2-byte SCI symbol.

Throughput measurements are obtained by summing the data received by all nodes for a fixed period of time, and dividing the total number of bytes received by the period of time, yielding a total effective throughput for the entire system. This throughput does not include any overhead due to packet headers, idle symbols, echos, etc. The total offered load is divided up equally among the nodes and sent to randomly distributed destinations. Since a responseless, 64-byte *move* operation was used, only one-way latencies are measured. The *move* subaction was selected in order to maximize the data throughput of the network. The latency of a packet is measured from the time the packet is generated and placed on the output queue to the time it reaches the destination node and is removed from the input queue.

Three sets of experiments were performed on the SCI model to gauge its performance under a variety of network topologies. First, uni- and bi-directional ring topologies were simulated to verify the models using the simpler, more easily predictable, 1D configurations. Next, 2D uni- and bi-directional tori topologies were constructed to study SCI under a more scalable environment. Finally, a uniform-ring-size torus topology that provides better fault tolerance is studied to compare its performance to a standard bi-directional torus.

2.2.1 One-dimensional topologies

The first set of simulations consists of single and counter-rotating ring topologies. Prior to conducting the simulations, there is a need to verify the accuracy of the results. In order to do so, the analytical best-case throughputs were calculated for both the single and counter-rotating ring topologies.

For the single-ring case, assuming a random distribution of data, each packet on a unidirectional SCI ring will traverse an average of *N/2* links before arriving at its destination. Using this value and the packet sizes mentioned above, the analytical best-case peak effective throughput (PET) estimate for an SCI ring using a store-and-forward approximation can be found by:

$$PET = \frac{(N\ nodes) \times (64\ bytes)}{\left(41 \times \left(\frac{N}{2}\right) + 5 \times \left(N - \frac{N}{2}\right) symbols\right) \times (2\ ns / symbol)}$$

The numerator of this equation represents the amount of data that is transmitted simultaneously, 64 bytes × *N* nodes. The denominator calculates the amount of time required for all *N* transactions to complete. This time is split into the time required for the 41-symbol request packets (40-symbol request packet and one idle symbol) to arrive at their destinations halfway around the ring and the time required for the 5-symbol *echo* packets (4-symbol *echo* and one idle symbol) to return. As expected the

peak throughput is independent of ring size and has a value of 1.39 GB/s.

By adding a counter-rotating ring, higher throughputs and lower latencies can be achieved because all links between nodes are now bi-directional. A packet sent on such a network can take the shortest path to its destination. To approximate an upper bound on the throughput of a counter-rotating ring network, the unidirectional peak effective throughput calculation can be modified as:

$$PET = \frac{(N\ nodes)\times(64\ bytes)\times(2\ packets)}{\left(41\times\left(\frac{N}{4}\right)+5\times\left(N-\frac{N}{4}\right)symbols\right)\times(2\ ns/symbol)}$$

resulting in a constant throughput of 4.57 GB/s.

In a counter-rotating ring, each node can transmit two 64-byte data packets at the same time, one on each ring. The numerator is therefore modified to allow twice the number of packets transmitted simultaneously. The denominator reflects the fact that request packets taking the shortest path will traverse *N*/4 links on average for a uniform distribution. Again, the peak throughput is independent of ring size. However, the value of the peak throughput is less than the ideal of 4 times that of a unidirectional ring. This drop in the peak performance is attributed to the longer distance that an *echo* packet must travel to return to the originating node.

Figure 3 shows the effective throughputs obtained from simulation experiments for both the single and counter-rotating ring topologies. The total effective throughput of any size unidirectional ring approaches 1.35 GB/s at saturation. For the counter-rotating ring topologies, the larger topologies saturate at approximately 3.5 GB/s. The 4-node counter-rotating ring saturates at a lower throughput than the other topologies because the assumption that packets travel an average of *N*/4 links is inaccurate for the 4-node case. Since nodes cannot transmit to themselves, the average number of links a packet must traverse is actually $N^2/[4(N-1)]$ for even *N*, or *(N+1)/4* for odd *N*. Both expressions approach *N/4* for large *N*. This effect is also the cause for the lower values of 6-, 8-, and 10-node throughput than the predicted value.

Table 2 compares the simulation results on average latency for the unidirectional and counter-rotating rings at light loads. The measurements at light loads were taken at an offered load of 0.6 GB/s which is far below the saturation point for either topology. As expected, the latencies for the counter-rotating systems are smaller than single-ring systems of equal dimensions.

Table 2. Unidirectional and counter-rotating ring one-way latencies at an offered load of 0.6 GB/s

System	Single-ring Latency (µs)	Counter-Rotating Ring Latency (µs)
4-node	0.176	0.132
6-node	0.225	0.146
8-node	0.282	0.164
10-node	0.344	0.184

The counter-rotating ring systems provide a higher throughput and lower latency than the single-ring systems. However, a limitation with both of the two simple ring systems presented above is that the total effective throughput is limited and does not scale as a function of the number of nodes. Moreover, the latencies continue to increase as the number of nodes increases. In order to achieve higher throughputs and lower latencies, other topologies using multiple rings in two dimensions are considered next.

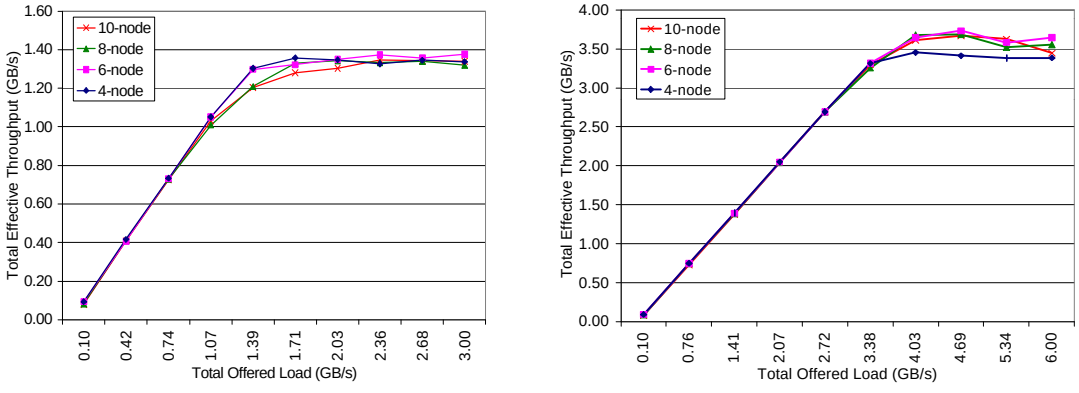


Figure 3. Unidirectional and counter-rotating ring throughputs

2.2.2 Two-dimensional topologies

In order to judge the performance of SCI under a more scalable, two-dimensional topology, networks of various sizes were constructed using uni- and bi-directional tori.

Figure 4 shows a 3x3 unidirectional 2D torus and a 3x3 bi-directional 2D torus. The 2D tori scale better than simple, 1D rings since adding more nodes to a torus also involves adding more rings. The torus structure allows for a more even distribution of the load and should provide lower latencies and higher throughputs.

In order to determine an analytical expression for the maximum throughput in a two-dimensional, unidirectional torus, consider a *k x k* torus with $N=k^2$ total nodes. The same approach used previously can be applied if the average number of hops for a uniform distribution is known. To find this average distance, the sum of the distances from an arbitrary starting node to each of the other *N-1* nodes must be found. If the starting node is chosen to be *(0,0)* at the lower-left node in a system such as that shown in Figure 4a, then it can be shown that the distance to any node *(i,j)* is *i+j*. Therefore, the average distance to any of the *N-1* nodes is:

$$Ave.\ Dist. = \frac{\sum_{i=0}^{k-1}\left[\sum_{j=0}^{k-1}(i+j)\right]}{N-1} = \frac{\sum_{i=0}^{k-1}\left[\sum_{j=0}^{k-1}(i+j)\right]}{k^2-1} = \frac{k^2}{k+1}$$

For the *echo* packets to travel from node *(i,j)* back to node *(0,0)* on a unidirectional ring requires *(k-i)+(k-j)* hops. Substituting this into the above equation yields exactly the same result. Using this equation for average distance and taking a similar approach as for the counter-rotating ring case, the peak effective throughput for a unidirectional 2D torus can be calculated as:

$$PET = \frac{(N\ nodes)\times(64\ bytes)\times(2\ packets)}{\left(41\times\left(\frac{k^2}{k+1}\right)+5\times\left(\frac{k^2}{k+1}\right)symbols\right)\times(2\ ns/symbol)}$$
$$= (\sqrt{N}+1)\times(1.39\ GB/s)$$

Unlike the single-ring cases, the peak throughput in a unidirectional 2D torus scales with $N^{1/2}$. As the number of nodes in the torus increases, so does the total available bandwidth. Table 3 shows the analytical best-case peak throughputs for various sized tori. The throughput results from the simulation experiments of the unidirectional 2D tori are shown in Figure 5a.

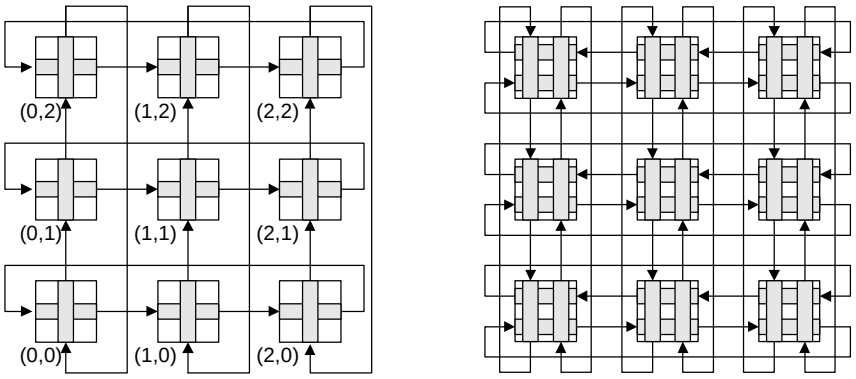
For the unidirectional tori, the simulated peak throughputs are consistently lower than the predicted values but follow the same numerical difference pattern between adjacent throughputs as the number of nodes increases. The reason the simulated throughputs are smaller than the analytical values is that there is a possibility for busy-retries to occur in a 2D torus that are not accounted for in the analytical model. Busy-retries can occur when many packets on a ring try to switch through the same node to a second ring. The input queues of the switch node may become full while the node waits for available bandwidth on the second ring. The resultant busy-retries on the first ring represent wasted bandwidth and reduce the throughput from the theoretical maximum.

Table 3. Analytical and simulated peak throughput with the unidirectional 2D torus

System	Analytical Peak Throughput (GB/s)	Simulated Peak Throughput (GB/s)
9-node	5.56	5.10
16-node	6.95	6.21
25-node	8.34	7.54
36-node	9.73	8.67

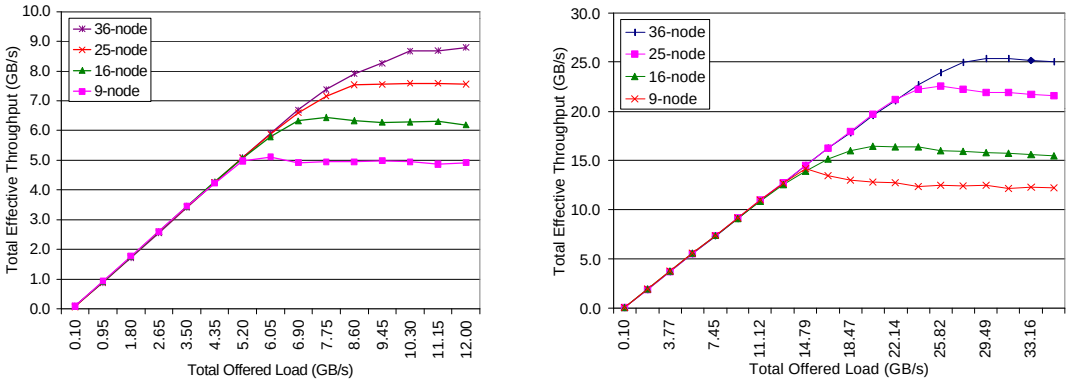
For the bi-directional tori, analytically, the throughputs might be expected to be approximately four times that of the unidirectional tori due to twice the output ports at each node and half the average distance that each packet must now travel. Again this is not the case as seen by the bi-directional tori throughput plots in Figure 5b. The difference is attributed to *echo* packets always taking the non-optimal path by continuing on the same ring as the request packets, as well as additional busy-retries. The additional busy-retries result from the fact that every node now has two input ports from one counter-rotating ring which may contend for access to the same port on the second counter-rotating ring. The problem is exacerbated by the fact that, like *echo* packets, busy-retries must continue on the same ring as the request packet, taking a non-optimal path to the source node.

A comparison of the latency results from the simulation experiments with lightly loaded counter-rotating rings and unidirectional tori are shown in Table 4. The latencies are measured at an offered load of 0.6 GB/s for both topologies. Even though the diameters of the tori are smaller than the diameter of the counter-rotating ring topologies, the tori exhibit a higher latency. This increase in latency is



(a) (b)

Figure 4. 3x3 unidirectional torus and bi-directional torus



(a) (b)

Figure 5. Unidirectional and bi-directional 2D torus throughputs

attributed to the additional switching delay not encountered by packets in the counter-rotating ring topology.

Table 4. Counter-rotating ring and unidirectional 2D tori one-way latencies at an offered load of 0.6 GB/s

System	Counter-Rotating Rings Latency(µs)	Unidir. Torus Latency (µs)
8-node	0.164	NA
9-node	0.170	0.206
10-node	0.184	NA
16-node	0.221	0.241

Table 5 compares the latencies of the unidirectional ring torus to the bi-directional 2D torus for light loads. The latencies of the bi-directional tori are lower than the unidirectional tori since the average distances between nodes is halved.

In order to demonstrate the flexibility of the model, the uniform-ring-size topology in Figure 6 was constructed and simulated. The shaded areas represent the SCI interfaces. In this example, each node connects to four different rings with a fixed size of four interfaces per ring. This topology maintains a constant ring size even though the number of nodes continues to increase. In contrast, an *N*-node torus using the topology described previously would use a ring size of $N^{1/2}$.

Table 5. Unidirectional and bi-directional 2D torus latency at a total offered load of 0.6 GB/s

System	Unidir. Torus Latency (µs)	Bi-dir. Torus Latency (µs)
9-node	0.206	0.162
16-node	0.241	0.197
25-node	0.293	0.212
36-node	0.312	0.222

By using a uniform ring size, the system becomes more fault tolerant. With smaller ring sizes, the number of rings in the system is larger for large *N*.

Since a faulty link renders an entire ring unusable, the smaller the ring size, the smaller the number of affected nodes. The price of such fault tolerance is a reduction in performance due to the additional switching time required to reach the final destination.

Figure 7 shows the throughputs measured for 9-, 16-, 25-, and 36-node uniform-ring tori and compares them to the throughputs of the bi-directional tori. The comparison shows a negligible decrease in throughput due to the additional switching. This result is expected since a packet would not have to take any more hops on average to reach its destination on the uniform-ring topology than on a bi-directional torus.

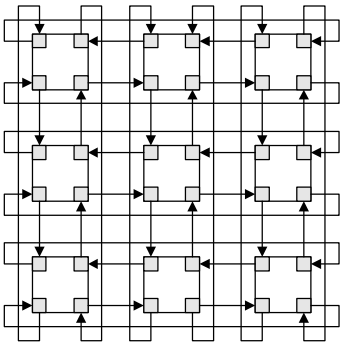


Figure 6. Uniform-ring-size torus

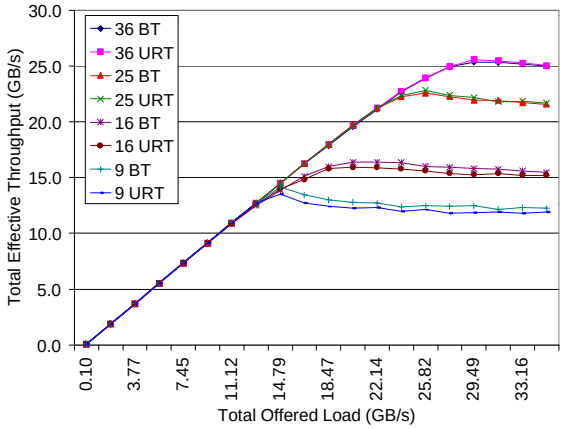


Figure 7. Bi-directional and uniform-ring-size torus throughputs

Table 6 compares the latency results from simulation experiments with both topologies. The latency at light loads is greater in the case of the uniform-ring-size torus due to the additional switching required to reach the destination. In the 9-node uniform-ring case, the latency is slightly lower that the 9-node torus since each ring traverses four nodes instead of three. The slight performance degradation of uniform-ring-size tori can be mitigated with the use of fast switches. The fault-tolerant aspects of the network may outweigh the performance requirements in some situations.

Table 6. Bi-directional and uniform-ring-size torus latency at an offered load of 0.6 GB/s

System	Bidir. 2D Torus latency (µs)	Uniform-Ring-Size 2D Torus latency (µs)
9-node	0.162	0.142
16-node	0.197	0.202
25-node	0.212	0.233
36-node	0.222	0.277

3.0 SCI fault-injection simulations

The fault-tolerant SCI model is based on the fault-detection protocols defined by the standard. These protocols use a passive form of fault detection where testing of the network is performed only in the event that a fault has occurred. Several system-level diagnosis algorithms exist that are adaptable to specific and arbitrary network topologies but many rely on active testing for fault-tolerance. Rangarajan [13] presents a distributed system-level algorithm for arbitrary topologies that uses periodic testing of neighboring nodes. This algorithm is the basic premise behind several such strategies presented in [14-16] that follow an active method of fault detection but assume that the interconnect does not fail. Much of the work was pioneered by Preparata, Metze, and Chien in [17]. Several protocols do exist that guarantee resilient communication between nodes in the event of interconnect faults. Dolev et al. [18] presents an end-to-end communication protocol based on source routing that uses sequence numbers attached to the messages to determine whether or not the paths between the source and destination nodes are "clean". Other methods of end-to-end communication protocols typically use flooding to achieve their goals in the presence of failures.

In order to detect, locate, and recover from failures, certain additions to the switch/node model are required. The additions are made in the form of a diagnostic/error handler. The handler consists of multiple sub handlers, one per SCI interface at each node. Since each SCI interface also has a unique subset of CSR registers, and recovery packets could reach the node through many paths, there is a need to share the information between the interfaces to aid in the detection and recovery from failures in the shortest time possible. The diagnostic/error handler for a 4-port switch is illustrated in Figure 8. It consists of three sub handlers per port. These include

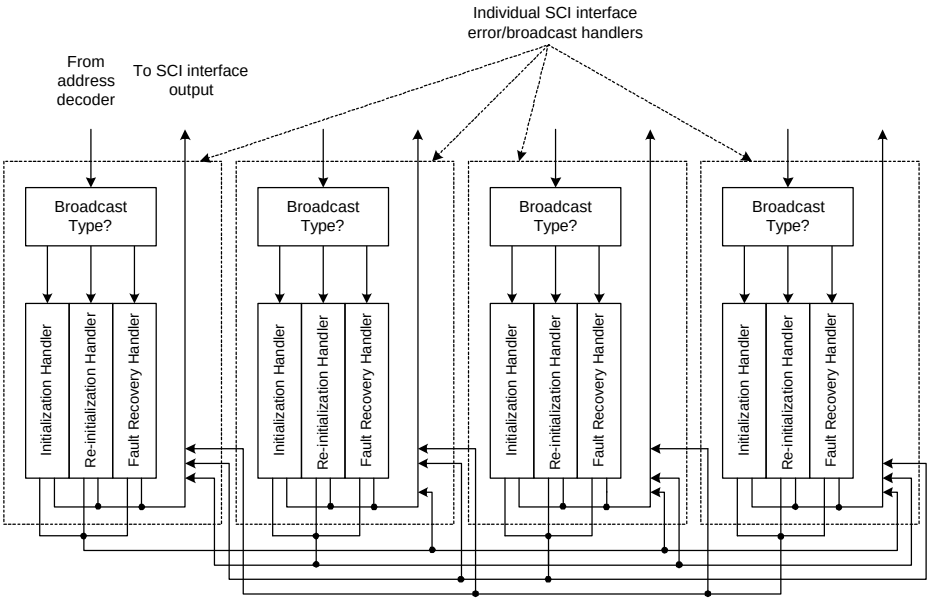


Figure 8. Diagnostic/error handler

an initialization handler that deals with initialization packets and generates the routing tables, a broadcast handler that ensures the progression of diagnostic packets and the elimination of duplicate ones, and a fault-recovery handler.

The fault-recovery handler is responsible for recovery from link failures, link stuck-at faults, inoperative processing units, insane processing units, inoperative SCI interfaces, insane SCI interfaces, inoperative switches, and routing faults within the network. Each fault-recovery handler is further subdivided into three units, one to recover from link failures, another to recover from switch and processing node failures, and a third used to recover from routing failures. The three sub units are shown in Figure 9.

The fault handlers operate on diagnostic packets whose commands are described later in this paper. By identifying the incoming diagnostic packet, each of the three fault-recovery units follow a set of predefined steps to isolate and recover from the identified network failure.

3.1 Fault-tolerance protocols

The fault-location and fault-recovery protocols developed are geared towards locating faults in the network rather than in the processing unit at each node, hence redundancy is only implemented in the network components. Limited coverage is provided for inoperative and insane processors by identifying and logging errors such that the remaining operational processors no longer send packets to the failed ones.

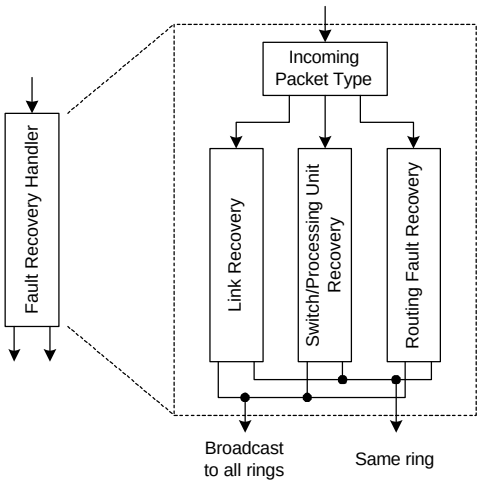


Figure 9. Fault-recovery handler

The targeted network faults include broken links, link stuck-at faults, inoperative switches and interfaces, and faults in the switch logic resulting in lost or misrouted packets. The goal is to provide the maximum possible level of fault detection, location, and recovery without compromising compatibility with the SCI standard.

We have opted not to include the fault-location and fault-recovery protocols in this paper, but instead describe the new commands and the additional registers defined to implement them. These protocols will be the subject of a future paper.

The fault-recovery protocols are based on two request types, *request-to-diagnose* packets and *request-to-recover* packets. *Request-to-diagnose* packets are targeted to specific nodes with instructions to carry out a certain type of diagnostic test. *Request-to-recover* packets on the other hand are issued when a fault has been diagnosed and located. They carry direct commands describing what operation must be performed by each node. An example of this would be a re-initialization command. The primary purpose of these packets is to permit error handlers at one node to access resource information at remote nodes for diagnostic purposes.

The formats of the *request-to-recover* and *request-to-diagnose* packets are the same as a 16-byte *smove* and 16-byte *dmove request-send* packets, respectively. The *command* field defines a 16-byte *move* operation and the 16 most-significant bits of the *address offset* are used to identify the function of the packet. In order to comply with the standard, user-defined commands such as the *request-to-diagnose* and *request-to-recover* commands are implemented by using the *address offset* bits instead of defining new commands. The new address offsets are summarized in Table 7.

3.2 Summary of new registers, flags and vectors

Two additional flags are defined in the bus-dependent portion of the STATE_CLEAR and STATE_SET registers. The SINK bit is set in order to sink all incoming packets, and a BYPASS bit prevents any packets from being removed by the SCI interface. The final form of the STATE_CLEAR and STATE_SET registers is shown in Figure 10.

In addition to the modifications made to the STATE registers, additional control and status registers are defined. These include the CRC_PARITY_ERROR_COUNT register, the PROCESSOR_QUEUE_LENGTH register, and the INPUT_QUEUE_ LENGTH register at offsets 388, 392, and 396 respectively of the bus-dependent address space as defined by the CSR architecture standard [19]. The addition of the above flags and registers does not alter the operation of SCI as defined by the standard.

Several vectors at each node are also required to monitor node resources in addition to other information such as retried packets, request timeouts, and failed processors. These vectors are defined in Table 8.

3.3 Fault-injection simulation results

The fault-injection simulations are carried out for four cases, link failures, processing unit failures, switch failures, and switch routing errors. All other failures are subsets of these failures and use the same recovery procedures.

Table 7. Diagnostic and recovery packet address offsets

Request-to-recover Packet Commands	
Command	**Address Offset (LSB)**
Re-initialize Request	aaaa00010000rrrr
Processor Fault Recovery	aaaa00100000rrrr
Routing Fault Recovery	aaaa00101000rrrr
Request-to-diagnose Packet Commands	
Command	**Address Offset (LSB)**
Processor Diagnosis	aaaa00011000rrrr
Switch Fault Diagnosis	aaaa00110000rrrr

rrrr – reserved bits
aaaa – other unused LSB

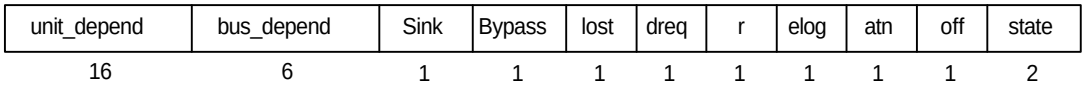


Figure 10. Final format of the STATE_CLEAR and STATE_SET registers

Table 8. Additional log vector definitions

Vector Name	Description
Transaction Timeout	Contains the targetID corresponding to the transactionID that timed out
Node Timeout Count	Keeps count of how many times a node has timed out
Insane Node Count	Keeps count of insane node transmissions
Echo Retry	Keeps count of retries of packets on the ringlet

3.3.1 Link fault simulations

Link faults are injected using a BONeS block that is inserted in links between nodes. The block is triggered at a time predefined by the user at the start of the simulation and can either stop the transmission of packets and idle symbols, or trigger a CRC fault in packets. Three topologies were tested in this study, 9-, 16-, and 25-node bi-directional tori. The goal was to induce link failures and incrementally degrade the topologies to unidirectional tori. The results were then compared to the performance simulation results of fault-free tori to determine the effectiveness of the fault-recovery methods and how the degradation affects system performance.

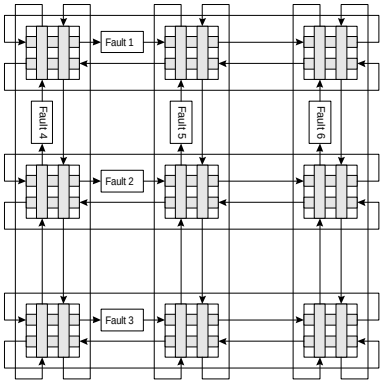


Figure 11. 9-node bi-directional torus showing order of injected faults

The topologies were degraded using the pattern illustrated in Figure 11 for a 9-node torus. The numbered links indicate the order in which the link faults are injected. Figure 12 shows the sampled values of throughput taken during the simulation. The total offered load is set such that each topology is saturated to allow meaningful comparison with the maximum throughputs obtained from the fault-free simulations. As can be seen from the plot in Figure 12, the systems recover from each injected link fault. Each link failure is represented by a dip in the graph where the faults occur followed by the recovery period. The plot does not present an accurate graph of the throughputs since the measured throughputs are averaged over a small period of time. Figure 13 shows a more accurate plot of the throughput vs. number of faults. The measurements for this graph start at the normal operation time of the network and end at the point the faults occur. It does not take into account the propagation time of the *request-to-recover* packets or the time required to re-initialize the network, thus providing a more accurate reading of the throughput.

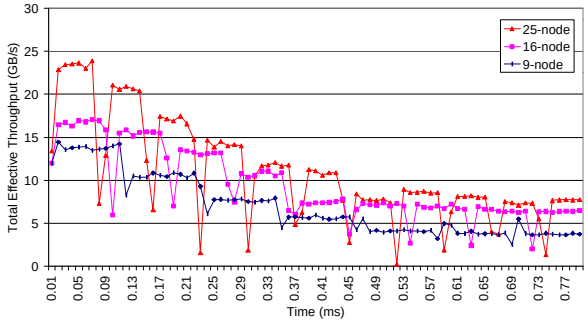


Figure 12. Bi-directional torus sampled performance with injected link faults

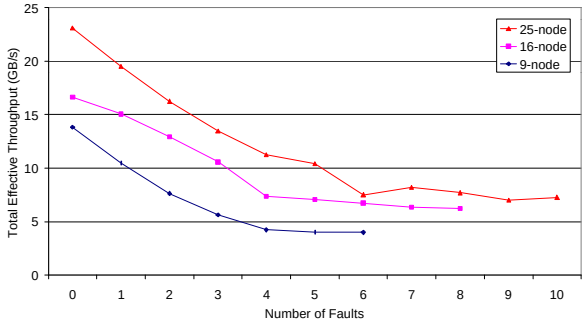


Figure 13. Bi-directional torus non-sampled performance vs. number of faults

A comparison of the fault-injection simulations to the fault-free simulations in Figure 5 verifies the initial and final values of the throughputs. All topologies exhibit the throughputs of bi-directional tori at the start of the simulations and the throughputs of the unidirectional tori at the end of the simulation period.

The intermediate values demonstrate the level of degradation in performance with an increasing number of link faults. The plots for the 9-node and 16-node systems show a continually degrading performance. In the 25-node case a dip after six faults is seen followed by a rise. This sudden drop is attributed to heavy congestion on the remaining horizontal rings after the five counter-rotating horizontal rings have been eliminated. The drop is not evident in the smaller topologies since the time to reach the destination is shorter and hence dropped packets do not affect the throughput measurements over the finite simulation period.

3.3.2 Switch failure simulations

Switch failures are simulated by disabling the arbiter within the targeted switch's crossbar. Doing so prevents switching requests from being processed by the arbiter and hence none of the input queues are granted access to the switch. Figure 14 demonstrates the systems ability to recover from switch failures. Each dip in the plot represents a failure and the time to recover from the failure. Since the values of the throughput are sampled over small periods of the total simulation time, the recovery time cannot be accurately measured from the graphs. Four switch failures were simulated for the 16-node torus and five for the 25-node torus. The failures for both topologies are set to occur at the same instant of time. The total offered load for both systems is 4 GB/s, which is below the saturation point of both topologies. The simulations show that the time to recover for the 25-node torus is longer than the 16-node torus which is as expected.

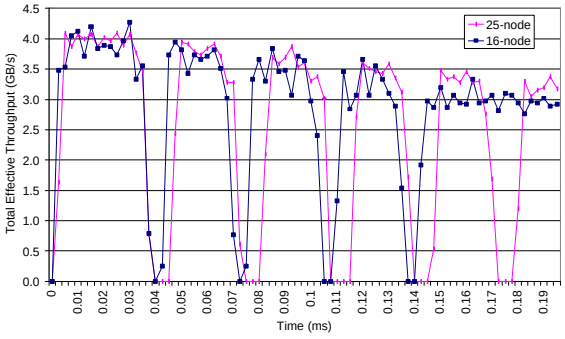


Figure 14. 16-node and 25-node tori with switch failures

Since each switch failure isolates the processing unit at the node and since the 4 GB/s total offered load is distributed evenly among all nodes, the total effective throughput for the 16-node plot drops by 250 MB/s per switch failure. Similarly the total effective throughput for the 25-node system drops by 160 MB/s for each switch failure.

3.3.3 Processor failure simulations

Processor failures are simulated by eliminating all outgoing requests from and sinking all incoming requests to the traffic generator. The result is an apparent failed processor. The nodes sending requests to the failed processor timeout and send *request-to-diagnose* packets to the faulty node. Figure 15 shows the sampled values of a 16-node and 25-node torus with failed processing units. Since the failures do not require the systems to re-initialize, the dips in the graph where the failures occur are not as obvious as in the failed switch plot. Arrows have been included in the plot to show the approximate location of the failures.

Again the performance degradation after each failure in the 16-node case is approximately 250 MB/s for a total offered load of 4 GB/s as each processor contributes 250 MB/s of the total offered load. The resulting final throughput after four processor failures is approximately 3 GB/s as shown in Figure 15. In the 25-node case, each processor failure reduces the total throughput by 160 MB/s resulting in a final throughput of approximately 3.2 GB/s after five failures.

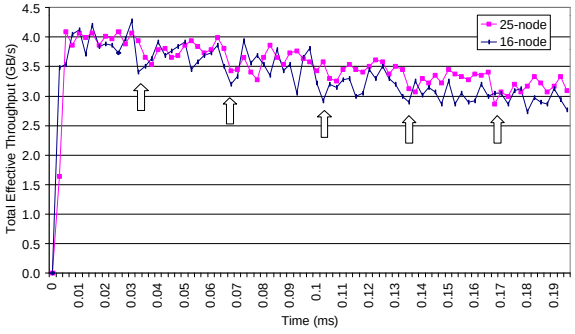


Figure 15. 16-node and 25-node tori with processor failures

4.0 SCI reliability simulations

Once the system matching the required performance parameters has been constructed, the reliability of the system must then be determined to ensure it matches the reliability requirements for the desired mission time. Determining the reliability of a networked system poses one major difficulty. As links within the network fail, each failure creates critical links that must remain operational to maintain node connectivity. These link interdependencies result in a large number of operational states that make classic reliability modeling with Markov chains impractical for SCI distributed switching fabrics. In order to

model systems with such large state spaces using Markov, modeling tools based on Petri nets are employed. Petri nets are used for modeling systems where events occur asynchronously or concurrently. They use a directed graph model whose nodes are divided into two sets called *places* and *transitions*. Transitions represent events and places represent the conditions for the events. For more information on modeling with Petri nets, the reader is referred to [20-24].

For our purposes, a Stochastic Activity Net (SAN) based package called UltraSAN was used. Unlike standard Petri net packages, UltraSAN defines two additional components, namely *input* and *output* gates that can be defined with predicates and functions to determine the markings of the model places after certain activities have fired. The function of these gates can be modeled in other packages using additional instant-time activities and places which simply complicate the final model. UltraSAN's implementation yields a far less complicated model. More detailed descriptions of the UltraSAN modeling environment can be found in [25-26].

Modeling in UltraSAN can be divided into three distinct steps. The first step involves defining the subnets that make up the model and then assembling the subnets into a *composed* model. Once the *composed* model has been completed, the global variables defined in the subnets are initialized and the studies based on the values of the global variables are defined. Finally the reduced base model state space is generated and solved using one of the analytic or simulative solvers provided by UltraSAN

After creating the models for both the one-dimensional and two-dimensional topologies, the next step is to assign values to the timed activities. Based on the gate count of the LincChip-16 developed at Interconnect Systems Solution [27], the failure rate for the SCI interface chip can be estimated using $\lambda=\pi_Q\pi_L(C1\pi_T+C2\pi_E)$. This formula is defined by the MIL-HDBK-217 Standard [28] to estimate the failure rate of digital integrated circuits. In addition, the estimated failure rates of the support chips to implement the error handlers can be estimated along with the failure rates of the connectors and PC boards [29]. Since the exact specifications of the circuits required to implement the system are not known, the values presented in Table 9 represent estimates of the failure rate for each interface, error handler, connector, and PC board combination.

In addition to this failure rate for SCI links, the failure rate for the crossbar is estimated at one failure per 10^6 hours [19]. Although these failure rates are estimates, keeping them constant throughout the modeling process allows for accurate comparison of the reliabilities of different systems. Using the UltraSAN SCI models developed, the reliabilities of both one- and two-dimensional SCI *k*-ary *n*-cubes were evaluated.

4.1 One-dimensional topologies

The simulated reliabilities for the single- and dual-ring topologies over a mission time of 10,000 hours are illustrated in Figure 16. It is found that larger ring sizes lower the overall system reliability. This reduction in reliability is due to the fact that all the SCI interfaces and their supporting components must be operational for the ring to remain operational. The reliability of the single-ring systems can be verified analytically by assuming a series system of links and crossbars, and can be expressed as

$$R_{system}(t)=R_{link}^{\ n}(t)R_{crossbar}^{\ n}(t)$$

where $R_{link}(t) = e^{-l_i t}$ and $R_{crossbar}(t) = e^{-l_c t}$ for $l_i = 3.509\text{x}10^{-6}$ failures per hour and $l_c = 1\text{x}10^{-6}$ failures per hour. The analytical reliabilities of the single-ring systems are shown in Table 10. The simulation results from the plot in Figure 16, and the analytical results from Table 10 are identical, verifying the correctness of our model and providing a baseline to compare the remaining simulations against.

The dual-ring simulation results, also shown in Figure 16, demonstrate the same pattern seen in the single-ring simulations with the smaller ring systems providing higher reliabilities that the larger ones. Comparing equally sized single and dual-ring system results, it can be seen that adding an additional ring improves the system's reliability considerably.

Table 9. Approximate failure rate per link

Component	**Estimated Failure Rate / 10^6 hours**
SCI interface (30K+ gates)	1.625
Error handler (10K-30K gates)	1.090
Connectors (2)	0.056
PC Board (commercial spec. quality)	0.738
Approximate total failure rate per link	**3.509**

Table 10. Analytical reliability estimates of single-ring SCI systems

Mission time (hours)	4-node reliability	6-node reliability	8-node reliability	10-node reliability
0	1.000	1.000	1.000	1.000
1000	0.982	0.973	0.965	0.956
2000	0.965	0.947	0.930	0.914
3000	0.947	0.922	0.897	0.873
4000	0.930	0.897	0.866	0.835
5000	0.914	0.873	0.835	0.798
6000	0.897	0.850	0.805	0.763
7000	0.881	0.827	0.777	0.729
8000	0.866	0.805	0.749	0.697
9000	0.850	0.784	0.723	0.666
10000	0.835	0.763	0.697	0.637

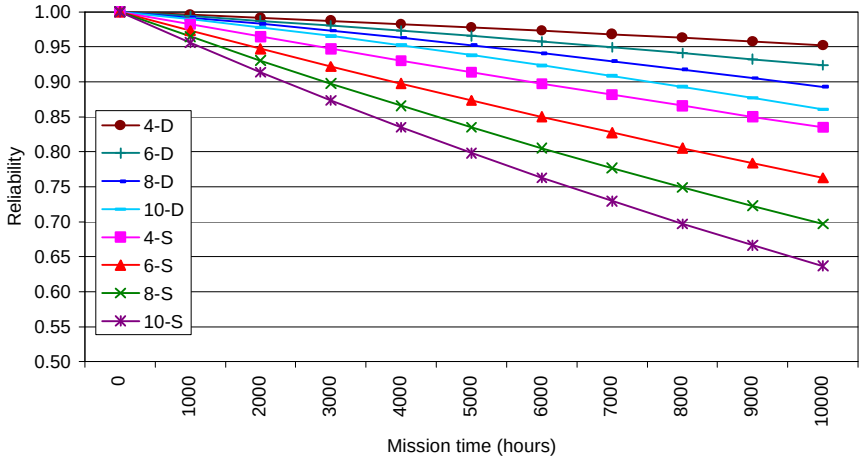


Figure 16. Reliabilities of single- and dual-ring SCI systems

The percentage improvement in the reliabilities of the dual-ring systems in comparison to the single-ring systems is shown in Table 11. The results show that adding the additional ring to the single-ring topologies provides a greater improvement in the reliability for the larger topologies than the smaller ones, for all mission times. The percentage increase in reliability follows a linear pattern, increasing with respect to time for a constant topology size. At constant mission times, the percentage increase in reliability also follows a linear pattern as the topology size increases, doubling each time the number of nodes double.

4.2 Two-dimensional topologies

The reliabilities for the unidirectional and bi-directional tori topologies are shown in Figure 17. As the sizes of the topologies increase, their reliabilities decrease. Again reliability is affected by the size of the ringlets making up the topology. The larger the ring size, the lower the system's overall reliability.

Comparing unidirectional and bi-directional tori with equal numbers of nodes reinforces the earlier conclusion that adding multiple rings while maintaining a constant number of nodes improves the reliabilities of the systems. Figure 17 shows the reliability of the 9-node unidirectional torus exceeding the reliability of both the 25-node and 36-node bi-directional tori for all mission times. It also shows the reliability of the 16-node unidirectional torus to be higher than that of the 36-node bi-directional torus for a mission time less than 2000 hours. These results represent two examples where topology size versus the number of nodes required by an application becomes an important factor in determining the desired system reliability.

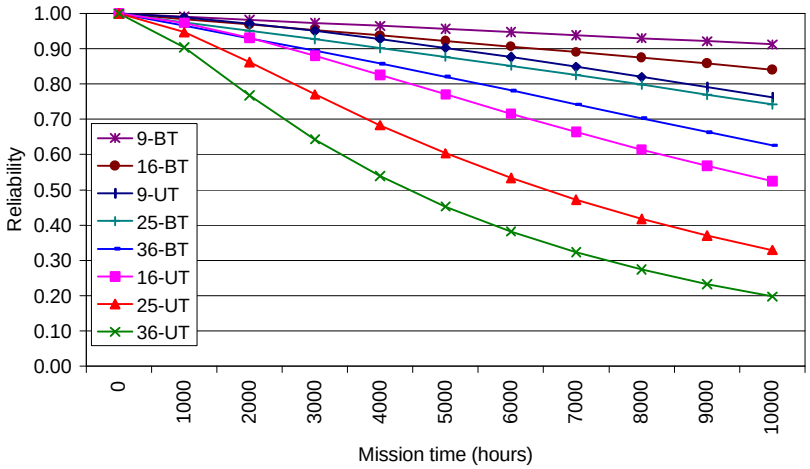


Figure 17. Unidirectional and bi-directional tori reliabilities

Table 11. Percentage improvement in the reliabilities of the dual-ring systems vs. the single-ring systems

	Percentage improvement in reliability for dual-ring systems vs. single-ring systems			
Mission time (hours)	**4-node**	**6-node**	**8-node**	**10-node**
1000	1.40	2.11	2.81	3.50
2000	2.80	4.22	5.62	7.02
3000	4.20	6.31	8.43	10.53
4000	5.62	8.43	11.22	14.04
5000	7.00	10.53	14.04	17.54
6000	8.44	12.65	16.84	21.06
7000	9.82	14.75	19.67	24.56
8000	11.24	16.85	22.46	28.07
9000	12.65	18.95	25.27	31.59
10000	14.05	21.06	28.08	35.09

Table 12. Percentage improvement in reliability for the unidirectional tori vs. the bi-directional ring tori

	Percentage improvement in reliability for bi-directional tori vs. unidirectional tori			
Mission time (hours)	**9-node**	**16-node**	**25-node**	**36-node**
1000	0.30	1.10	3.06	6.73
2000	1.06	3.97	10.35	21.00
3000	2.31	8.21	20.27	39.01
4000	3.88	13.50	32.08	59.26
5000	5.83	19.66	45.26	81.19
6000	8.06	26.57	59.57	104.62
7000	10.61	34.16	74.83	129.71
8000	13.42	42.33	90.91	156.51
9000	16.43	51.08	107.71	185.59
10000	19.69	60.22	125.20	216.96

The percentage improvement in the reliabilities of the bi-directional tori in comparison to their unidirectional counterparts is shown in Table 12. It demonstrates an increasing pattern for all topology sizes, as was the case of the percentage improvements in the reliabilities of the one-dimensional systems. The reliabilities of the 1D systems are easy to predict relative to topology size since the percentage increase in reliability follows a linear pattern, doubling each time the size of the topology doubles. For the tori, the percentages increase at a more rapid pace.

5.0 Conclusions and future research

In this paper we presented the latest results in an on-going research project to model, simulate, and evaluate fault-tolerant SCI-based distributed switching fabrics. The performance of multiple SCI switched topologies was examined analytically and through simulative analysis. It was shown how the performance of SCI can be made more scalable by using multi-dimensional, switched topologies. The performance of a uniform-ring-size torus was also examined to gauge the effect of additional fault tolerance on performance. Though the simulations presented in this paper were intended to represent computers connected via an SCI system-area network, the results could be applied on a multiprocessor level where the individual nodes represent processors, memory modules, I/O controllers, etc.

The fault-free simulation results illustrated several interesting points regarding the scalability of SCI. Though an ideal quadrupling in throughput would be expected as counter-rotating rings are added to the single-ring and tori cases, it was shown that the throughputs actually increase by only 2.5 to 3 times the throughputs of the unidirectional systems. The latencies at light loads show an improvement in the range of 1.6 to 1.8 for the counter-rotating ring and bi-directional tori systems in comparison to their unidirectional counterparts.

The model was also extended to incorporate fault-tolerance protocols through the definition of new control and status registers, the definition of new fault-recovery commands, and the incorporation of error handlers. Minimal changes were made to the reserved fields intended for future SCI extensions, preserving the operation of the standard. The protocols were designed to be general and independent of the switch fabric. In the test systems a crossbar-based switch was used, however the internal switch fabric does not play any role in the fault-tolerance protocols. The switch represents a single point of failure at the node that can be remedied using additional redundancy.

The simulations presented demonstrate each system's ability to recover from link failures, link stuck-at faults, switch failures, and processing node failures. All other failures such as insane nodes, insane interfaces, and routing faults are subsets of the simulated failures and follow the same recovery procedures. The performance of three bi-directional tori was measured as the systems were continually degraded to unidirectional tori by injecting link failures. The results demonstrated how the throughput is affected by link failures since bottlenecks can develop resulting in unexpected behavior. This bottleneck was most accentuated in the 25-node torus case and is expected to be more pronounced in larger tori.

Finally, we determined the reliability of several 1D and 2D SCI *k*-ary *n*-cubes. The results illustrated the extent to which the number of SCI interfaces per ring impacted the system reliability. The systems constructed using smaller rings demonstrated a higher reliability than those composed of larger rings.

The results showed the tradeoffs in designing for reliability in distributed switching fabrics for SCI. An example of such a tradeoff was illustrated in the 9-node unidirectional torus simulation where the system's reliability exceeded the reliability of both the 25-node and 36-node bi-directional tori for all mission times. It also showed the reliability of the 16-node unidirectional torus to be higher than that of the 36-node bi-directional torus for a mission time less than 2000 hours.

Future research in this area will target performance and fault-injection evaluations of 3D tori and other inherently redundant ring-based topologies. Internal switch architectures such as MINs, shared memory, and bus-based fabrics can easily be incorporated and evaluated within the model. Other fault-free aspects of SCI can also be examined including the effects of queue lengths and switching speeds on the performance of different topologies.

Currently work is being conducted at the HCS Research Lab to incorporate hardware-in-the-loop simulations permitting real parallel applications to execute and communicate over simulated networks. Once the model has been refined and modified to interface with the Integrated Simulation Environment (ISE) [12], fault-injection effects on parallel applications can then be studied. Such a system presents a challenge since additional policing is required to include checkpointing, reassignment, and rollback in the event of bisected networks. An example of this is a smart system using a master-

worker paradigm, where "backup" masters are now elected based on network conditions such as critical links instead of being elected randomly.

Acknowledgements

This work was sponsored in part by the National Security Agency. The authors also acknowledge contributions by Matthew Chidester, Robert Todd, and William Phipps in our lab for their assistance in the development of our high-fidelity CAD model for SCI.

References

[1] IEEE, SCI: Scalable Coherent Interface, *IEEE Approved Standard IEEE 1596-1992*, 1993.

[2] B. Wu, A. Bogaerts, R. Divia, E.H. Kristiansen, H. Muller, B. Skaali, "Several Details of SCI Switch *University of Oslo/CERN Internal Report*, November 1993.

[3] B. Wu, "SCI Switches," *Proceedings of the International Data Acquisition Conference on Event Building and Event Data Readout in Medium and High Energy Physics Experiments*, Fermilab in Batavia, Illinois, October 1994.

[4] B. Wu, A. Bogaerts, B. Skaali, "A Study of Switch Models for the Scalable Coherent Interface," *The Sixth Conference on Data Communication Systems and their Performance*, Istanbul, Turkey, October 1995.

[5] R. Divia, "SwitchLink V1.0," *CERN DRDC RD24 Project Internal Note*, November 1992.

[6] T. Hou, S.R. Tsai, L. M. Tseng, "Adaptive and Fault-Tolerant Routing Algorithms for High Performance 2D Torus Interconnection Networks," *Computers and Applied Math*, vol. 23, no. 1, pp. 3-15, 1992.

[7] L. Gravano, D. Pifarre, P. Berman, J. Sanz, "Adaptive Deadlock and Livelock-Free Routing with all Minimal Paths in Torus Networks," *IEEE Transactions on Computers*, vol. 5, no. 12, pp. 1233-1251, December 1994.

[8] A. Chien, J. H. Kim, "Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors," *JACM*, vol. 42, no. 1, pp. 91-123, 1995.

[9] B. Fang, X. Li, W. Wang, "An Optimal Fault-tolerant Routing Algorithm for Dual-Ring Networks," *International Symposium of Fault-Tolerant Computing, Digest of Papers*, pp. 192-195, 1989.

[10] R. Gould, Graph Theory, *The Benjamin/Cummings Publishing Company, Inc*., Menlo Park, California, 1988.

[11] K. Shanmugan, V.S. Frost, and W. LaRue, "A Block-Oriented Network Simulator (BONeS)," *Simulation*, Vol. 58, No. 2, pp. 83-94, February 1992.

[12] A. George, R. Fogarty, J. Markwell, and M. Miars, "An Integrated Simulation Environment for Parallel and Distributed System Prototyping," *Simulation*, to appear.

[13] S. Rangarajan, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 312-333, February 1995.

[14] F. Meyer, D. Pradhan, "Dynamic Testing Strategy for Distributed Systems," *IEEE Transactions on Computers*, vol. 38, no. 3, pp.356-365, March 1989.

[15] R. Bianchini, R. Buskens, "Implementation of On-Line Distributed System-Level Diagnosis Theory," *IEEE Transactions on Computers*, vol. 41, no. 5, pp. 616-626, May 1992.

[16] S. Hakimi, K. Nakajima, "On Adaptive System *IEEE Transactions on Computers*, vol. 33, no. 3, pp. 234-240, March 1984.

[17] F. Preparata, G. Metze, R. Chien, "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Transactions on Computers*, vol. EC-16, no. 6, pp. 848-854, December 1967.

[18] S. Dolev, J. L. Welch, "Crash Resilient Communication in Dynamic Networks," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 14-26, January 1997.

[19] IEEE, Standard for Control and Status Registers Architecture for Microcomputer Buses, *IEEE 1212-1994*, 1994.

[20] C. Lindemann, "DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets," *Proceedings of the Sixth International Conference on Modeling Techniques and Tools for Computer Systems Performance Evaluation*, pp. 15-29, Edinburgh, Great Britain, 1992.

[21] G. Chiola, "GreatSPN 1.5 Software Architecture," *Proceedings of the Fifth International Conference on Modeling Techniques and Tools for Computer Systems Performance Evaluation*, Torino, Italy, 1991.

[22] G. Ciardo, J. Muppala, and K. S. Trividi, "SPNP: Stochastic Petri Net Package," *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*, pp. 142-151, Kyoto, Japan, December 1989.

[23] N. Lopez-Benitez, J. A. B. Fortes, "Detailed Modeling and Reliability Analysis of Fault-Tolerant Processor Arrays," *IEEE Transactions on Computers*, vol. 41, no. 9, pp. 1193-1200, September 1992.

[24] C. Beounes et al., "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems," *Proceedings 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23), IEEE*, Toulouse, France, June 1993.

[25] W. H. Sanders, L. M. Malhis, "Dependability Evaluation Using Composed San-Based Reward Models," *Journal on Parallel and Distributed Computing*, vol. 15, no. 3, pp. 238-254, July 1992.

[26] W. H. Sanders, W. D. Obal, M. A. Qureshi, F. K. Widjanarko, "The UltraSAN modeling Environment," *Performance Evaluation*, vol. 24, pp. 89-115, 1995.

[27] K. Kibria, Interconnect Systems Solution, http://www.iss-us.com/LincCore.htm.

[28] MIL-HDBK-217F: Handbook for Reliability Prediction of Electronic Equipment, *Defense Printing Service*, Philadelphia, PA.

[29] W. Yost, "Cost Effective Fault Tolerance for Network Routing," Master of Science Thesis, *University of Washington*, 1995.