PERFORMANCE ANALYSIS AND FAULT TOLERANCE OF RANDOMIZED ROUTING ON CLOS NETWORKS

Manjit Bhatia
Department of Computer Science
Bowie State University
Bowie MD 20715
e-mail: mbhatia@cs.bowiestate.edu
and
Abdou Youssef
Department of EE and CS
The George Washington University
Washington D.C 20052
e-mail: youssef@seas.gwu.edu

ABSTRACT

Beside universality and very low latency, Youssef's randomized self-routing algorithms [25] have high tolerance for multiple faults and more strikingly have the potential for fault tolerance without diagnosis. In this paper we study the performance of Youssef's routing algorithms for faulty Clos networks in the presence of multiple faults in multiple columns with and without fault detection. We show that with fault detection and diagnosis, randomized routing algorithms provide scalable, very efficient and fault tolerant routing mechanisms. Without fault detection and diagnosis, randomized routing provides good fault tolerance for faulty switches in either the first or the second column. The delays become large for faults in the third column or for faults in more than one column. In conclusion, randomized routing enables the system to run without periodic fault detection/diagnosis, and if and when the performance degrades beyond a certain threshold, diagnosis can be performed to improve the routing performance.

Keywords : Clos Networks, Fault Tolerance, Permuting, Randomized Routing.

1 Introduction

Network fault tolerance [2, 7, 17, 22, 24] is essential to the reliability and performance of parallel computer systems. This is especially true in the case of multistage interconnection networks (MIN), which have more components (e.g. switches), and thus experience more faults, than static networks.

Network fault tolerance involves fault detection, fault diagnosis, and fault accommodation. Fault detection in networks is often resolved by periodic system checking or by the mechanism of acknowledgments of message delivery and time-out; both methods incur high overhead. Fault diagnosis of MIN's, which received considerable attention [4, 8, 10, 16, 19, 21, 23] but only for MIN's of small switch sizes (2 X 2 or 4 x 4), is extremely costly in time and/or hardware when switch sizes are large and when multiple faults are to be tolerated. Therefore, it is very desirable to have the ability to tolerate faults either without fault detection and diagnosis, or with diagnosis - but without explicit fault detection - performed only when the performance degradation crosses a certain threshold.

As for fault accommodation, several approaches have been proposed. In the case of omega-like banyan MIN's, the two main approaches are (1) the hardware approach of adding extra switches and/or extra links [1, 3, 13], and (2) the software approach such as multiple routing passes through the network [4, 18, 22]. These approaches incur hardware and/or latency penalty. In the case of Benes networks [5], several fault-tolerant routing schemes have been proposed [3, 12, 24] which take advantage of the multiple paths or add some extra hardware. However, these schemes do not solve the basic problem of the slow Benes routing control: They either route a subclass of permutations at a fast control speed or route all permutations at a very slow control speed. To remedy this situation, we proposed in [25] randomized self-routing algorithms for Benes and Clos networks [9], and studied some of their fault tolerance capabilities in [6]. These algorithms, applicable to all permutations and to asynchronous routing, were shown to deliver high fault-free and fault-tolerant performance.

This paper is a continuation of our work on fault-tolerant randomized routing on Clos networks cited above. The significant new contributions of the present paper are two-fold. Firstly, we obtain performance analysis results for the routing delays in the presence of multiple faults in multiple columns of Clos networks, as a function of network size and the number of faulty switches. Thus, we remove the restriction in [6] where faults were assumed to be in a single column only. Secondly, we show that one of our randomized routing algorithms in circuit-switched Clos networks tolerates switch faults without explicit fault detection and without fault diagnosis. This paper presents performance analysis of this very desirable capability.

Our performance analysis shows that with fault detection and diagnosis, randomized routing algorithms provide scalable, highly efficient and fault-tolerant routing mechanisms even in the presence of multiple faults in multiple columns. As the number of faulty switches increases, the average delay increases slowly and the Clos network degrades gracefully. In the absence of fault detection and diagnosis, we find that randomized routing provides good fault tolerance for faulty switches in the first or the second (but not both) column. Also for this case (i.e. when faulty switches are either in the first or the second column), the average delay increases slowly as a function of the number of faulty switches, and network degradation is quite gradual. However, the delays increase dramatically for faults in the third column or for faults in more than one column. In these cases, as the number of faulty switches increases, the average delay increases rapidly and the Clos network degrades quickly. In summary, the performance analysis presented in this paper leads us to conclude that randomized routing enables the system to run without periodic fault detection/diagnosis, and if and when the performance degrades beyond a certain threshold, diagnosis can be performed to improve the routing performance.

The rest of the paper is organized as follows. The following section gives a brief overview of Clos networks and discusses related routing concepts. The details of the fault models assumed in this paper are specified in Section 3. A review of the routing algorithms is presented in

Section 4. Section 5 discusses issues of static connectivity and dynamic full access that arise in Clos networks in the presence of multiple faults in multiple columns. Details of the performance analysis are presented in Section 6. Section 7 contains a summary of our work, conclusions and some ideas for future work.

2 Overview of Clos Networks

In this section, we present a brief overview of Clos networks and related routing concepts.

Throughout this paper p and q denote two positive integers and N = pq. A Clos network C(p, q) (see Figure 1) has N input terminals representing processors, N output terminals representing the same processors, and three columns of switches (left, middle, and right). Each of the left and right columns has p crossbar switches of size q X q. The middle column has q crossbar switches of size p X p. The switches in the left and right columns are labeled 0, 1, ..., p-1, and the switches in the middle columns are labeled 0, 1, ..., q-1. The input ports and the output ports of every n X n switch have local labels 0, 1, ..., n-1 (n = p or q). The inter-column connectivity is as follows: The y-th output of the x-th switch in one column is linked to the x-th input of the y-th switch in the next column. The input (or output) y of a switch x in any column

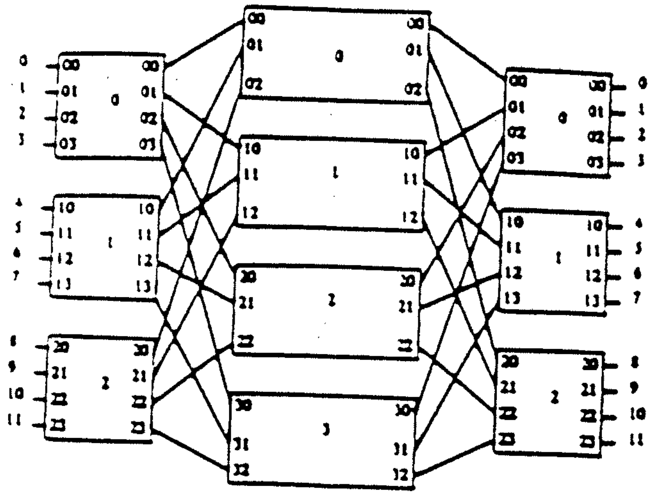

Figure 1: A Clos Network

has a global label [xy]. Every switch x in the left, middle or right column will be denoted $[x]_{left}$, $[x]_{middle}$, or $[x]_{right}$, respectively. Similarly, every input/output port [xy] of the left, middle or right column will be denoted by $[xy]_{left}$, $[xy]_{middle}$, or $[xy]_{right}$, respectively. Figure 1 displays Clos network C(3,4).

It is well known that any message at the output port $[xy]_{left}$ can be self-routed to any destination address $[x'y']_{right}$. This can be seen as follows. Because the output port $[xy]_{left}$ is connected to the input port $[yx]_{middle}$, the message enters the port $[yx]_{middle}$. It then uses the first digit x' of the destination address [x'y'] to exit through the output port$[yx']_{middle}$. Because this port is in turn linked to the input port $[x'y]_{right}$, the message enters the switch x' in the right column. It now uses the second digit y' of the destination address [x'y'], and exits through the output port $[x'y']_{right}$. Thus to reach the destination [x'y'], x' is used to set the middle column switches and y' to set the right colúmn switches.

The central idea of our randomized routing is as follows. For each input port $[xy]_{left}$ in a switch x in the left column, choose ***randomly*** an output port z in switch x and let the message exit through the output port $[xz]_{left}$. From the output port $[xz]_{left}$ self-routing takes over and the message reaches the destination $[x'y']_{right}$ as described above. Thus, the randomly chosen integers z, $0 \leq z \leq q-1$, control the switches in the left column just as the two digits x' and y' in the destination address [x'y'] control the switches in the middle and right columns, respectively. Because the output ports in the left column are chosen randomly, conflicts over links between the columns are bound to arise and have to be resolved. We will use unbuffered circuit switching and resolve conflicts as follows: Whenever two or more paths conflict over one link, only one path is fully established while all others are abolished awaiting future network cycles. (A network cycle is the time interval needed to (1) establish non-conflicting source-destination paths, and (2) deliver the corresponding messages.)

3 The Fault Model

Our fault model is the stuck-at model where a faulty switch is stuck at a one-to-one setting. The justification for this fault model is the following. An unbuffered n X n switch is (or can be) implemented in VLSI as an n X n grid of 2 X 2 switches. It is known that 2 X 2 switches can be stuck at one of their two states, cross or straight-through. Thus if certain 2 X 2 switches in an n X n crossbar switch are stuck, then certain input ports of the crossbar can link to some output ports but not to others. In the case where fault diagnosis is periodically performed, it is too costly in time and space to keep track of the set of accessible output ports for each input port in each faulty switch, and to adjust the routing algorithm accordingly; on the other hand, if we assume the switch is stuck at some one-to-one setting realizable by the faulty switch, the necessary routing adjustment is more manageable. However, when no fault diagnosis is needed, there is no need to reduce the accessibility information to a one-to-one setting, as we will discuss later. Nevertheless, we keep the one-to-one assumption for now because it simplifies our simulations.

The fault model assumes multiple switch faults. However, if switch faults occur in different columns, certain destinations may become inaccessible from certain sources, and multiple passes through the network are then required. We discuss the connectivity issues in Section 5.

4 The Randomized Routing Algorithms for Clos Networks

Two self-routing algorithms, namely, the *single randomization algorithm*, and the *multiple randomization algorithm* were presented in [6]. In the single randomization algorithm, once an input terminal selects (randomly) an output port of the first column switch, it continues to try to establish the path through the same output port in all its attempts until the routing of the permutation is completed. On the other hand, in the multiple randomization algorithm, each input terminal selects (randomly) *a new* output port of the first column switch in each attempt to establish a path to its destination. It was shown in [6] how these algorithms can tolerate switch faults in the case where diagnostic information about faulty switches is available at every input terminal. We will not discuss this matter here; the reader is referred to [6] for further details.

The matter of significance is that multiple randomization naturally tolerates faults in circuit-switched Clos networks <u>without</u> fault detection and diagnosis. This is because when a source S tries to establish a path to a destination D, if the path cannot be established because it is blocked due either to conflict or to a stuck switch, then the partially established path is abolished and the source S randomly selects a new control tag which is very likely to be different from the current path. As long as there is at least one establishable path between S and D, this path will eventually be chosen with probability equal to 1 asymptotically. It is clear that a source need not know the reason why its path did not go through (whether because of conflict or because of stuck switches). The repeated randomization enables the sources to try different paths until a fault-free path is established.

The following observations about the diagnosis-free multiple randomization routing algorithm should be noted:

- The number, positions, and settings of the stuck switches affect the latency of messages but not the routing algorithm itself, so the routing algorithm need not be modified.

- The switches may or may not be assumed to be able to "tell" if they cannot link a given input port to a requested output port (because of conflict or internal faults). If stuck switches do not know that they cannot honor requests correctly, requests are misrouted and paths may be established to the wrong destinations. This is handled by simply having the wrong destination signal back to the source (using the circuit-switching feedback wires) that it is the wrong destination; the source tries again a new path using a new random control tag. This maintains the correctness of routing without diagnosis.

- The stuck-at configurations need not be one-to-one. Correct routing is maintained as long as there is an establishable path between every source-destination pair. More generally, the fault model can be any model, whether of the stuck-at variety or not, as long as (1) there is a mechanism by which sources are notified that their paths cannot be established, and (2) there is at least one establishable path between every source-destination pair. However, for reasons mentioned earlier, we assume in our simulations that the faults are stuck-at one-to-one settings.

5 Multiple Faults and Connectivity of Clos Network

It can be easily seen that if we have faulty switches in two or more columns, then full connectivity of the network is lost. In other words, in case of multiple faults in multiple columns, there exist permutations that can not be routed in a single pass. Under these circumstances, the permutations can still be routed in multiple passes provided we have dynamic full connectivity. Note that in multiple pass routing, messages are sent to intermediate destinations before they reach their ultimate destinations. In the remainder of this paper we consider randomized routing with multiple passes and outline below the additional steps necessary in our routing algorithms.

For any given fault configuration of the network, we compute the graph G= (V, E), called the *connectivity graph*, where the vertices V are the set of N (=q^2) processors and E is the set of edges:

E = { (i, j) | a (direct) path can be established from input terminal
i to output terminal j }

This graph G can be obtained from the fault configuration of the network by the algorithm outlined below:

Algorithm construct_connectivity_graph G;

1. Construct a directed graph G' that models faulty Clos C(p, q) as follows. Each input terminal is represented by a node, and so is each output terminal. Each unfaulty switch is represented by a single node, and each link entering or leaving that switch is represented by an edge entering or leaving the corresponding node. Finally, each faulty (nXn) switch (where n = p or q) is represented by an n X n

directed bipartitite graph (V_1, V_2, A) where V_1 (resp., V_2) represent the n input ports (resp., output ports) of the switch, and

A = {(i,j)| input port i is stuck linked to output port j}.

Every link entering or leaving input i (resp., output i) of the faulty switch is represented by an edge entering (resp., leaving) node i of V_1 (resp., V_2).

2. For each input terminal node i, perform a breadth-first search BFS(G', i) from node i to determine which output terminals are reachable from i. Then construct the set of edges:

E = {(i, j)| output terminal node j is reachable from node i by BFS(G',i)}.

end construct_connectivity_graph G;

If the above graph G is fully connected (i.e., every pair of nodes are adjacent), then we have *static* connectivity in the network. If the graph G is strongly connected (but not necessarily fully connected), then we have *dynamic full accesibility in the network.* Deciding full connectivity is simply checking if G is complete, and deciding strong connectivity is a well known procedure based on depth-first search [20]

Before using our routing algorithm in the presence of faults, we first test dynamic full accessibility by constructing the connectivity graph G and by checking its strong connectivity. Any faulty configuration for which we do not have dynamic full accessibility is discarded in our simulation study. In case of dynamic full accesibilty, the minimum set of intermediate nodes (destinations) between every source-destination pair of nodes need to be determined. This can be done by broadcasting G to all processors and letting each processor i perform BFS on G from node i. The resulting BFS tree gives the shortest paths (and hence the minimum set of intermediate nodes) from i to each destination.

6 Performance Analysis

We have carried out performance analysis (through simulation) of randomized routing on Clos networks in the presence of faulty stuck-at switches in one or more columns. We note that theoretical performance analysis of these algorithms for fault-free case was carried out earlier and reported in [6]. In this section we present the results of simulations for randomized routing in the presence of a number of different configurations of faulty switches in three columns of Clos networks. To be specific, we choose a Clos network C(q,q), $q= 32$. To investigate scalability, we will show comparison with smaller Clos networks also.

The following steps are carried out for different fault configurations and for various values of $q \leq 32$, and form a template to obtain the results for routing delay:

1) The location of the faulty switches, if any, in any desired column is chosen randomly. The 1-to-1 switch settings of the faulty switches are also chosen randomly.

2) With the above fault configuration of the network as input, the presence of dynamic full accessibility of the network is checked. If not present, this particular fault configuration is discarded and we go back to step 1.

3) Several hundred randomly selected permutations are routed. Each input terminal i attempts to establish a path to its destination terminal j in each of the permutations.

4) We record the total number of cycles needed to complete the routing of the N ($= q^2$) messages for each of the permutations. (We define the cycle to be the time interval needed to establish non-conflicting paths and deliver the corresponding messages.)

5) The average number of cycles per permutation over all the routed permutations is computed.

6) The maximum number of cycles required among all the permutations is obtained. The spread between the average and the maximum gives an indication of the predictability value of the "average" metric.

We will use the results for the fault-free case as our baseline and compare our results for faulty networks to this case. We consider first the performance of fault tolerance with diagnosis and then without diagnosis.

6.1 Performance of Fault Tolerance with Diagnosis:

Recall that each processor i has for each each destination j a minimum set of intermediate nodes from i to j. While routing permutations each input terminal i attempts to send its message to its destination j by sending the message to the next node in the i-to-j path where it is put in a queue behind other messages that might be waiting. In subsequent cycles, this process is repeated until all messages reach their intended destinations.

We have considered a variety of configurations of faulty switches in the three columns of Clos network. In Figures 2 through 9 we present the following results as being typical .

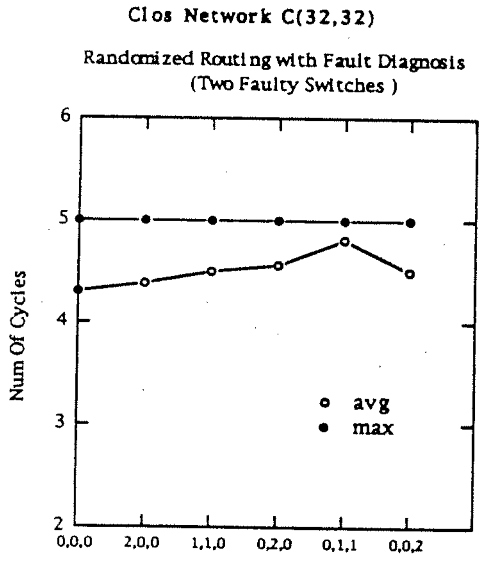

Figure 2

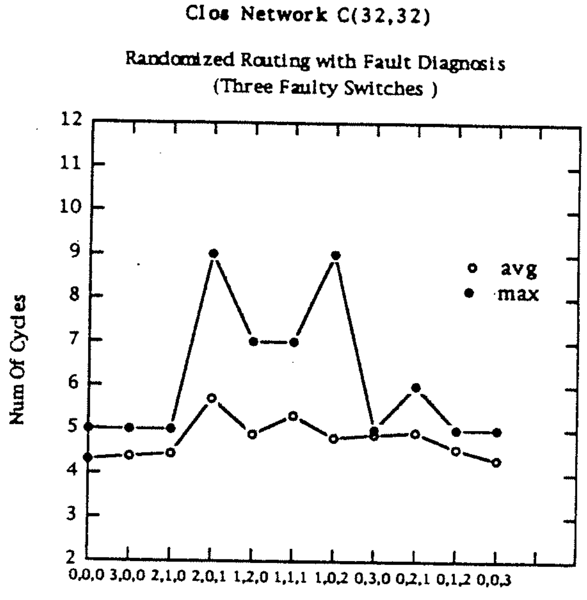

Figure 3

1) Two faulty switches in the network: We consider all possibilities for the locations of the two faulty switches. Both faulty switches could be in

any one column or any two of the columns. In Figure 2, the x-axis is labeled by 3-tuples to identify the location of the faulty switches. For example the tuple (1,0,1) means that there is one faulty switch in the first (left) column and one in the third (right) column and no faulty switches in the second (middle) column.

2) Three faulty switches in the network: We consider all (ten) possibilities for the locations of the faulty switches namely (3,0,0), (2,1,0), (2, 0,1), (1,2,0), (1,1,1),(1,0,2), (0,3,0), (0,2,1), (0,1,2), (0,0,3). (See the notation explained above.) The corresponding results are shown in Figure 3.The results for the above two scenarios are indicative of the dependence of the routing delay on the distribution of a fixed number of faulty switches in the three columns of the network.

First, we see from these graphs that in the presence of two faulty switches, anywhere in the network, the average delay is only one half cycle more than the fault free case. In the presence of three faulty switches, anywhere in the network, the average delay is only 1.4 cycle more than the fault free case. Secondly, these graphs demonstrate that the routing delay is fairly independent of the distribution of the faulty switches among the three columns of the Clos network.

These two observations lead us to infer that randomized routing exhibits excellent fault tolerance in the presence of a small number (up to three) stuck-at faulty switches *anywhere* in the network.

3) Next, we examine the behavior of the routing delay as the number of faulty switches grows steadily up to 25% of the total number of switches in the network. We summarize the results in Figure 4 by plotting the routing delay versus the number f of faulty switches, $1 \leq f \leq 8$, *in each column*. This graph gives us a sense of how the routing delay grows with increasing numbers of faulty switches in the network. We observe that for one faulty switch in *each column*, the average routing delay is 5.31 cycles and the maximum delay is 7 cycles. As the number of faulty switches increases, the average (resp., maximum) routing delay increases gradually to 75.06 cycles (resp., 98 cycles) for the case when there are 8 faulty switches in each column (i.e. 25% of all the switches in the network are faulty). This leads us to conclude that

the network would degrade gracefully as the number of faulty switches increases.

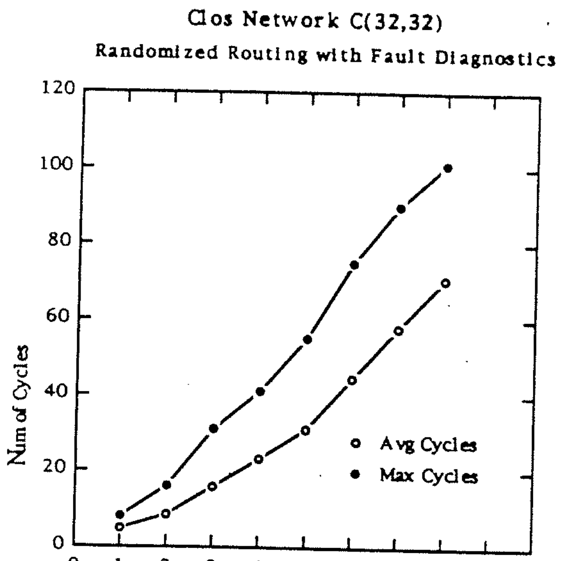

Figure 4

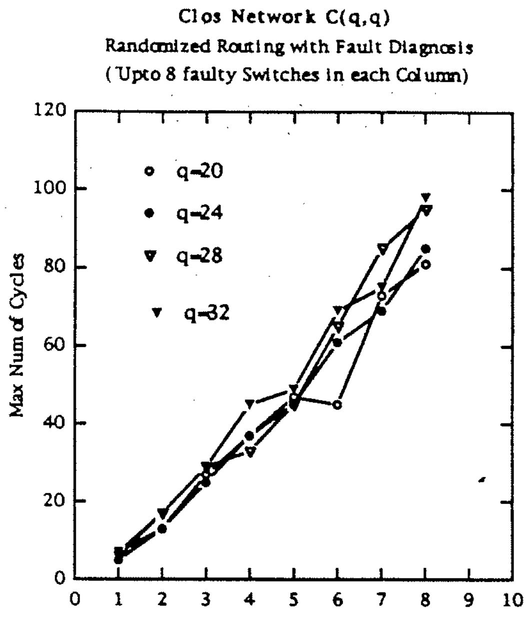

Figure 5

To examine the scalibility, in Figure 5 we compare the maximum routing delay as a function of number of faulty switches *f*, 1 ≤ *f* ≤ 8, *in each column*, in Clos networks of size q= 20, 24, 28, 32. The narrow band in which all the curves lie brings out the fact that the routing delay as a function of the number of faulty switches is quite independent of the network size. Thus, we have demonstrated that the good fault-tolerant characteristics of randomized routing scale very well with network size.

6.2 Performance of Fault Tolerance without Diagnosis:

In this subsection we present the results of randomized routing in the presence of faulty switches but without fault detection and diagnosis.

We first show that as long as there are establishsable paths between a given source-destination pair (s, d), the multiple randomization algorithm will establish one correct path with probability 1 asymptotically. We can readily prove the following theorem:

Theorem 1
In a faulty Clos network C(p, q) of N terminals ($N = pq$), let c be the number of establishable paths between a given source-destination pair (S, D) (Note that $0 \leq c \leq N/q$). In the presence of no other traffic, the probability that the randomization routing algorithm will take at most k attempts to establish a correct path S --> D is $1 - (1 - cq/N)^k$.

The above theorem has the following consequences:

- If there is at least one establishable path from S to D (i.e., $c \geq 1$), then $0 < \frac{cq}{N} < 1$, and thus the probability of taking k attempts to find the first establishsble path is strictly greater than 0.

- More importantly, since $1 - (1 - cq/N)^k$ goes to 1 as k increases, the probability of eventually selecting an establishable path is asymptotically 1. In other terms, S will certainly connect to D eventually.

- The number, positions, and actual configurations of the stuck switches affect the network performance only indirectly by way of affecting the number c of establishable paths between a given source-destination pair.

- The theorem quantifies the extent to which latency is affected by faults. For example, if a quarter of S--> D paths are disabled due to faults, then the probability of needing k attempts to find one establishable path is $1 - (1/4)^k$. In particular, the number of needed attempts is at most 4 with probability 0.99609.

Of course, the above probabilistic results apply when there is no traffic other than between S and D. In the presence of traffic and ensuing path conflicts, more attempts are needed to establish source-destination paths. However, the above theoretical results are highly indicative of the overhead caused by the lack of diagnostic information. To obtain more accurate estimates of the actual delay and the overhead associated with diagnosis-free multiple randomization, we have carried out simulations of C(32,32) under a variety of faulty switch configurations. We describe our results in the following.

When we study the performance of fault tolerance without fault detection or diagnosis, again we ensure that we have dynamic full accessibility in the network. However, the input terminal i is presumed to be unaware of the path it should take to reach its destination j. Under these circumstances, it is possible that an input port i can go into an infinite loop trying to establish a direct path to an output port j to which no direct path exists. This problem is resolved by keeping a count of the attempts to establish a direct i-to-j path, and after a predetermined number of attempts (typically 6), a path is allowed to be established between input terminal i and any available output terminal k in which the message is queued behind any other messages waiting to be routed to their respective destinations. Subsequently, the terminal k attempts to send this message to the intended destination port j. This process is repeated until all messages reach their intended destinations. In general, several passes are necessary before the entire permutation is routed. Results for the routing delay for different faulty switch locations are as follows.

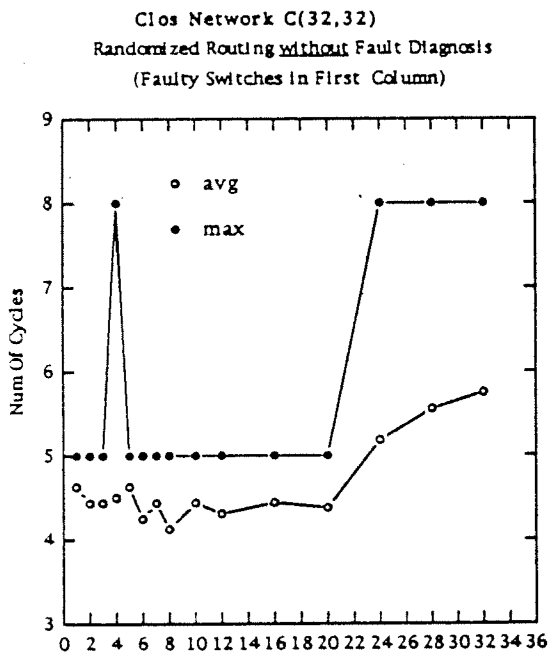

Figure 6

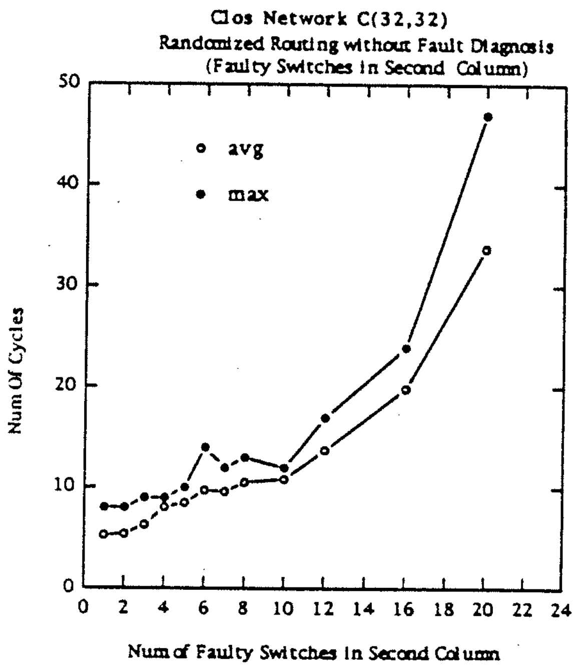

Figure 7

Faulty switches in the first (left) column only. Figure 6 depicts the routing delay as a function of the number of faulty switches in the first column. We observe that the average routing delay remains less than six cycles even when all the switches are faulty.

Faulty switches in the second (middle) column only. Results for this case are shown in Figure 7. We observe that the average routing delay remains less than eleven cycles for up to ten faulty switches anywhere in the second column. (Note that the locations of faulty switches, within the second column, and their settings are chosen randomly.) When the number of faulty switches in the second column grows to twenty faulty switches, the average routing delay increases to 34 cycles.

Faulty switches in the third (right) column only. As we see in Figure 8 the average routing delay is 179 cycles even for a single faulty switch in the third column. As the number of faulty switches in the third column increases, the average routing delay increases steadily to 336 cycles when we have ten faulty switches in the third column. This large delay will of course slow the system greatly.

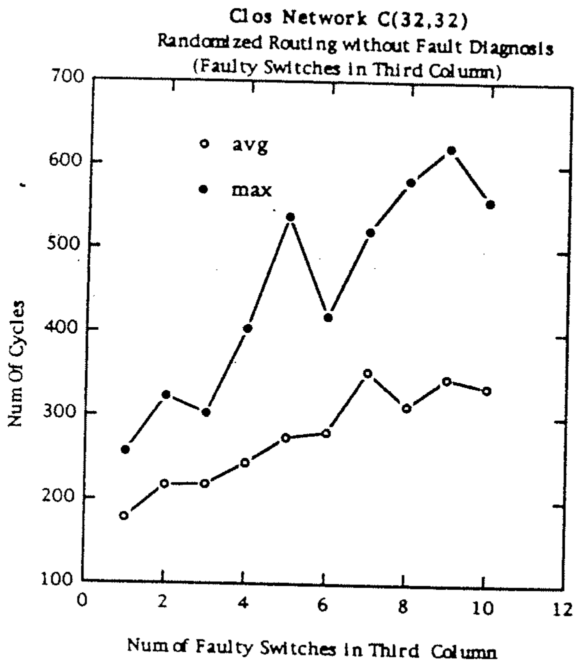

Figure 8

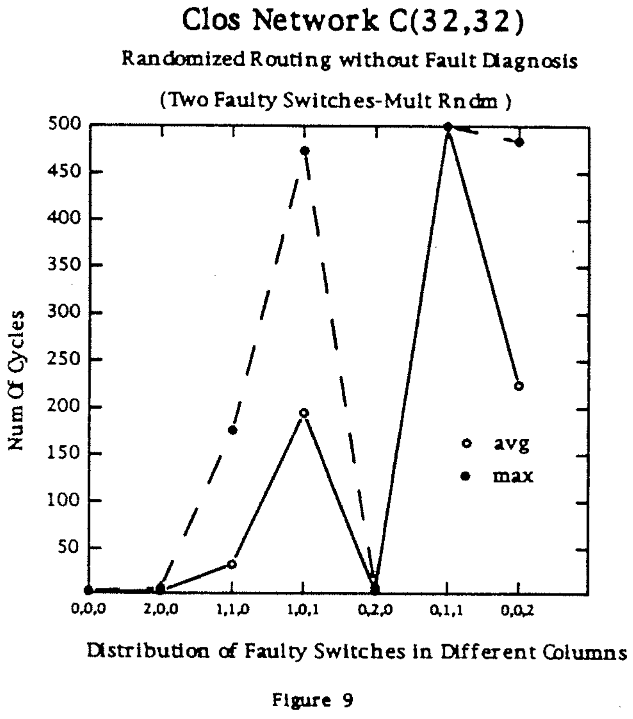

Figure 9

Two faulty switches in the network. The results for the routing delay are shown in Figure 9 as a function of the locations of the faulty switches in different columns (for explanation of the notation see the previous subsection). The wild fluctuation in the average routing delay for different locations of the faulty switches is quite evident. Note also the large delay when one or more of the switches is in the third column.

In the absence of fault detection and diagnosis, randomized routing provides good fault tolerance for faulty switches in first or second (but not both) columns. The delays become very large for faults in the third column or for faults in more than one column. As the number of faulty switches increases, the average delay increases rapidly and the Clos network would degrade quickly. A diagnostic test would then be highly desirable to improve the performance to acceptable levels.

7 Summary and Conclusions

In this paper we studied the fault tolerance capabilities of multiple randomization self-routing algorithm for Clos networks. It was demonstrated that with fault detection and diagnosis, randomized routing provides scalable, highly efficient and fault tolerant routing mechanisms; as the number of faulty switches increases, the average delay increases slowly and the Clos network degrades gracefully. In the presence of (say) three faulty switches, anywhere in the network, the average delay is only 1.4 cycles more than the fault-free case. As the number of faulty switches increase, the average delay increases gradually to 75 cycles when 25% of the switches are faulty.

In the absence of fault detection and diagnosis, randomized routing provides good fault tolerance for faulty switches in the first or second (but not both) column. The delays become very high (179 cycles for a single faulty switch) for faults in the third column or for faults in more than one column. As the number of faulty switches increases, the average delay increases rapidly and the Clos network degrades quickly.

Based on the above results of our performance analysis, we conclude that randomized routing enables the system to run without periodic fault detection/diagnosis for a variety of fault configurations. If and when the performance degrades beyond a certain threshold, diagnosis can be performed. This diagnostic information can be incorporated into our randomized routing thus leading to greatly improved routing performance. This fault tolerance scheme of performing diagnosis when degradation crosses a certain threshhold is certainly advantageous over periodic diagnosis because diagnosis is costly. The appropriate degradation threshhold can be determined based on the diagnosis cost and the performance gain achieved when fault information is available.

Future related work includes theoretical fault tolerance performance analysis, generalization to broader fault models where stuck switches can realize more than one one-to-one settings and other than one-to-one settings.

Accknowledgement: The second author wishes to thank Professors Miroslaw Malek and Rami Melhem for insightful discussions which inspired this work.

References

[1] G. B. Adam and H. J. Siegel, "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supercomputer", *IEEE Trans. Comput.*, Vol. C-31, No. 5, pp443-454, May 1982.

[2] G. B. Adam, D. P. Agrawal and H. J. Siegel, "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks", *Computer*, Vol. 20, No. 6, pp. 14-27, June 1987.

[3] D. P. Agrawal, "Testing and Fault Tolerance of Multistage Interconnection Networks", *Computer*, pp. 41-53, Apr. 1982.

[4] D. P. Agrawal and J. -S. Leu, "Dynamic Accessibility Testing and Path Length Optimization of Multistage Interconnection Networks", *IEEE Trans. Comput.*, C-34, pp.255-266, Mar. 1985.

[5] V. E. Benes, *Mathematical theory on connecting networks and telephone traffic*, Academic Press, New York, 1965.

[6] M. Bhatia and A. Youssef, " Efficient Randomized Fault-tolerant Routing on Clos Network," *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 217-224. July 1992.

[7] M. Chen and K. G. Shin, "Adaptive Fault-Tolerant Routing Routing in Hypercube Multicomputers", *IEEE Trans. Comput.*, Vol. 39, No. 12, pp. 1406-1416, Dec. 1990.

[8] V. Cherkassky, E. Opper and M. Malek, "Reliability and Fault Diagnosis Analysis of Fault Tolerant Multistage Interconnection Networks", *Proc. 14th Ann. Int'l Symp. on Fault-Tolerant Computing*, pp. 178-183, 1984.

[9] C. Clos, "A study of Non-blocking Switching Networks," *Bell System Tech Journal*, Vol. 32, pp.406-424, 1953.

[10] N. J. Davis IV, W. T.-Y. Hsu and H. J. Siegel, "Fault Location Techniques for Distributed Control Interconnection Networks", *IEEE Trans. Comput.*, Vol. C-24, pp.902-910, Oct. 1985.

[11] T. Feng and W. Young, " An O(log^2 N) Control Algorithm," *Proc. of the Int'l Conf. Par. Proc.*, pp. 334-340, 1985.

[12] S.-T. Huang and C.-H. Tung, "On Fault-Tolerant Routing of Benes Networks", *Journal of Information Science and Engineering*, Vol. 4, pp. 1-13, July 1988.

[13] V. P. Kumar and S. M. Reddy, "Augmented Shuffle-Exchange Multistage Interconnection Networks", *IEEE Computer*, pp. 30-40, June 1987.

[14] K. Y. Lee, " A New Benes Network Control Algorithm," *IEEE Trans. Comput.*, C-36, pp. 768-772, May 1987.

15] G. F. Lev, N. Pippenger and L. G. Valiant, "A fast Parallel Algorithm in Permutation Networks," *IEEE Trans. Comput.*, C-30, pp. 93-100, Feb. 1981.

[16] A. Mourad, B. Ozden and M. Malek, "Comprehensive Testing of Multistage Interconnection Networks", *IEEE Trans. Comput.*, Vol. 40, No. 8, pp. 935-951, Aug. 1991.

[17] D. K. Pradhan, "Fault-tolerant Multiprocessor Link and Bus Network Architectures," *IEEE Trans. Comput.*, Vol. 34, No 1, pp. 33-45, Jan. 1985.

[18] J. P. Shen and J. P. Hayes, "Fault-Tolerance of Dynamic Full-Access Interconnection Networks", *IEEE Trans. Comput.*, Vol. 34, No 1, pp. 241-248, Mar. 1984.

[19] S. Thanawastien and V. P. Nelson, "Obtimal Fault Detection Sequences for Shuffle/Exchange Networks", *Proc. 13th Ann. Int'l Symp. on Fault-Tolerant Computing*, pp. 442-445, June 1983.

[20] R. E. Tarjan, "Depth First Search and Linear Graph Algorithms", *SIAM J. Computing* 1:2, pp. 146-160,1972.

[21]N. -F. Tzeng, P.-C. Yew and C. -Q. Zhu, "Fault-Diagnosis in a Multi-path Interconnection Network", *Proc. 16th Ann. Int'l Symp. on Fault-Tolerant Computing*, pp. 98-103, 19836

[22] A. Varma and C. S. Raghavendra, "Fault-Tolerant Routing in Multistage Interconnection Networks," *IEEE Trans. Comput.*, Vol. C-38, No. 3, pp. 385-393, March 1989.

[23]] C. L. Wu and T. Y. Feng, "Fault-Tolerant Routing in Multistage Interconnection Networks", *IEEE Trans. Comput.*, Vol. C-30, pp.743-758, Oct. 1981.

[24] Y. -M. Yeh and T. -Y. Feng, "Fault-Tolerant Routing on a Class of Rearrangeable Networks", *Int'l Conference on Parallel Processing*, Vol. I, pp. 305-312, 1991.

[25] A. Youssef, " Randomized Routing algorithms for Clos Networks", *Computers & Electrical Engineering, an International Journal*, Vol. 19, No. 6, pp.419-429, 1993.