Efficient Randomized Fault-Tolerant Routing on Clos Networks

Manjit Bhatia

Department of EECS
The George Washington University
Washington, DC 20052

Abdou Youssef

Department of EECS
The George Washington University
Washington, DC 20052

Abstract – *This paper will propose and study two randomized self-routing fault-tolerant algorithms,* Single randomization*(SR) and* multiple randomization *(MR), for routing arbitrary permutations on Clos networks. The algorithms set the first column of Clos* **randomly** *and self-route the messages in the remaining 2 columns. When permuting with SR, every source randomly selects an output port of the first column, and then attempts to set up its path until it succeeds. With MR, whenever a source fails to set up its path due to conflicts, it randomly selects a* **new** *output port of the first column. We will show that the permutation delay in Clos of up to 1024 nodes is 3-6 time units under SR, and 3-4 time units under MR. It is also shown that the delay degradation due to multiple switch faults is less than one time unit when the faulty switches are the first or last column, and if the faulty switches are in the middle column, the delay degradation is at most 3 units under SR and at most 2 units under MR. The universality, high fault tolerance, self-routedness and very small routing delays make randomized routing superior to any non-randomized routing algorithm for Clos networks. They also make Clos networks very attractive for parallel computer systems.*

1 Introduction

Fault-tolerant interconnection networks are essential to the reliability of parallel computer systems. Much work has been done on network fault tolerance [2, 6, 11, 15, 20, 22]. In the case of multistage interconnection networks (MIN), most of the fault tolerance efforts have focused on the Ω-like banyan MIN's because of their self-routedness and lower cost. However, these networks have limited fault tolerance because when a switch is stuck, certain destinations become inaccessible from certain sources. Two approaches have been used to enhance the fault tolerance capabilities of these networks. The first is a hardware approach [1, 3, 11] of adding extra switches and/or extra links. The second is a software approach such as multiple routing passes through the network [4, 16, 20]. These techniques, though they improve the network fault tolerance, cause hardware and/or latency penalty.

Benes [5] and Clos networks [8] are more fault-tolerant because of the availability of multiple paths between sources and destinations. Several fault-tolerant routing schemes have been proposed for Benes networks [3, 10, 22] that 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 speed: They either route a subclass of permutations at a fast speed or route all permutations at a very slow speed. As for Clos networks, no specific fault tolerance scheme is known. This lack of attention to Clos networks may stem from (1) the higher hardware cost of Clos networks due to the required large switches and (2) the slow routing speed on Clos which takes $O(N \log^2 N)$ [12]. The hardware problem has been greatly alleviated by the advances of VLSI which make large crossbar switches affordable. The routing problem, though improved by certain recent approaches [23] for routing subclasses of permutations, remains a problem for routing arbitrary permutations. The problem of fast, fault-tolerant routing of arbitrary permutations on Clos networks is the focus of this paper.

We will propose and study fault-tolerant randomized self-routing algorithms for Clos networks. Randomized routing has been applied by Valiant on hypercubes [19] and by Mitra and Cieslak on the so-called extended Omega [13], but the fault tolerance potential of randomized routing was not addressed by those authors. We will give two randomized algorithms, *the Single Randomization Algorithm* and *the Multiple Randomization Algorithm*. The first algorithm is similar to Mitra and Cieslak's algorithm, but the second is an enhancement that will be shown to reduce the routing delay significantly. We will then show how these two algorithms easily adapt to multiple switch faults. Finally, we will conduct performance analysis of the routing delay of permutations under these two algorithms and in the presence of switch faults of a stuck-at type. The analysis will be carried out both theoretically and through extensive simulations.

Our performance analysis will show that the algorithms route any permutation in 3–7 network cycles with overwhelming probability, and that most faulty switches add to the delay at most one cycle on average. (A network cycle is the time needed to establish non-conflicting source-destination paths and deliver the corresponding messages.) The analysis will also show that multiple randomization yields better performance than single randomization.

The paper is organized as follows. Next section will give an overview of Clos networks. Section 3 will specify the fault model assumed in this paper. Section 4 will present the two randomized routing algorithms

0-8186-2870-7/92 $03.00 © 1992 IEEE

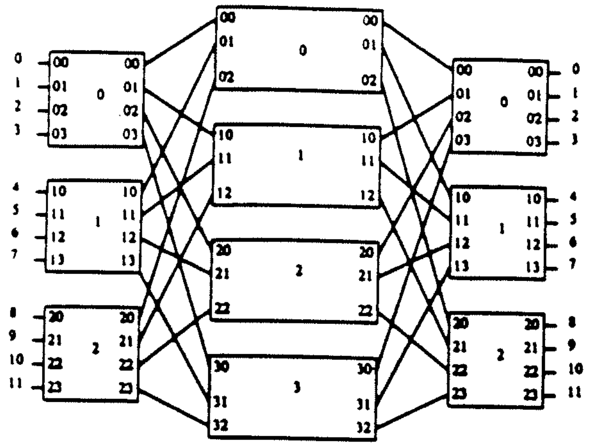


Figure 1: A Clos Network

and show how they can tolerate switch faults. Section 5 will give a probabilistic analysis of the communication delay of permutations in faulty Clos using randomized routing. Section 6 will present the simulation results of the communication delay. Concluding remarks will be presented in Section 7.

2 Overview of Clos Networks

Clos networks will be overviewed and related routing concepts discussed.

Let p and q be two positive integers and $N = pq$ throughout. A Clos network $C(p,q)$ (see Figure 1) has N input terminals representing processors, N output terminals representing either memory modules or processors, and three columns of switches (left, middle and right). Each of the left and right columns has p crossbar switches of size $q \times q$. The middle column has q crossbar switches of size $p \times p$. The switches in the left and right columns are labeled 0, 1, ... , $p-1$, and the switches in the middle column are labeled 0, 1, ... , $q-1$. The input ports and the output ports of every $n \times n$ switch have *local labels* 0, 1, ... , $n-1$ ($n = p$ or q). The input (or output) y of a switch x in any column 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/ouput port $[xy]$ of the left, middle or right column will be denoted $[xy]_{left}$,$[xy]_{middle}$, or $[xy]_{right}$. The interconnection between the first two columns links every output port $[xy]_{left}$ to input port $[yx]_{middle}$. Similarly, the interconnection between the middle column and the right column links output port $[yx]_{middle}$ to input port $[xy]_{right}$.

It is known that if the first column of $C(p,q)$ is dropped, the remaining network can be self-routed using the destination addresses. To see this, suppose that a message is to be sent from output port $[xy]_{left}$ of $C(p,q)$ to the output terminal $[x'y']$. The message first enters the input port $[yx]_{middle}$, then, using the digit x' of the destination address $[x'y']$, the message exits through output port $[yx']_{middle}$. Afterwards, it enters the input port $[x'y]_{right}$ and then, using the digit y' of the destination address $[x'y']$, it reaches output port $[x'y']_{right}$, which is the desired destination. In summary, to go to destination $[x'y']$, x' is used to control the middle column, and y' is used to control the right column.

Consequently, to go from any input terminal $[vw]$ of $C(p,q)$ to any output terminal $[x'y']$, we can select a digit z', $0 \le z' \le q-1$, and form the *control tag* $z'x'y'$ to find a path from $[vw]$ to $[x'y']$ in $C(p,q)$ as follows: Use z' to control the left column, that is, to link input port $[vw]$ to output port $[vz']$ of the first column, and from there establish the path to the output terminal $[x'y']$ as explained above. The way z' is selected characterizes the routing algorithm. In this paper, z' will be selected **randomly**.

Since z' is selected randomly, conflict over links is bound to occur and should be resolved. We will use unbuffered circuit switching and resolve conflicts as follows: Whenever two or more paths conflict while being established, only one path is fully established while all the others are abolished awaiting future cycles.

3 The Fault Tolerance Model

The three standard aspects of fault tolerance are (1) type of faults, (2) number of faults, and (3) detection and location of faults (fault diagnosis). Efficient techniques for switch fault diagnosis in MIN's are available [3, 7, 9, 14, 17, 18, 21], although these techniques are mainly for banyan MIN's with 2×2 switches. In any case, fault diagnosis in Clos, though of great importance, will not be considered in this paper. We will only specify the first two aspects of our fault model and assume that the detection of the faults and the identification of their locations and type are provided.

Our fault model is the *stuck-at model* wherein a faulty switch is stuck at a one-to-one setting. We will denote by α_x the setting (i.e., permutation) of a stuck switch x. The justification for this fault model is the following. An unbuffered $n \times n$ switch is (or can be) implemented in VLSI as an $n \times n$ grid of 2×2 switches. It is known that 2×2 switches can be stuck at one of their two states, *cross* or *straight through*. Thus, if certain 2×2 switches in an $n \times n$ crossbar switch are stuck, then certain input ports of the crossbar can link to some output ports but not to others. If we were to keep track of the set of accessible output ports for each input port in each faulty switch, with the intent to make that information available to the input terminals to adjust their randomizing accordingly, then the randomization adjustment overhead would likely outweigh the benefit, let alone the necessary memory overhead of storing that information. On the other hand, if we assume the switch is stuck at some one-to-one setting realizable by the faulty switch, the necessary randomization adjustment is more manageable.

The fault model assumes multiple switch faults. However, if switch faults occur in different columns, certain destinations become inaccessible from certain sources, and multiple passes through the network are then required. Algorithms to route from any source to any destination (using perhaps multiple passes) in a faulty banyan MIN have been introduced [4, 16, 20].

ıese algorithms may be extended to apply to Clos. ɒut since randomized routing is concerned with single-pass routing, these multiple-pass algorithms have no direct bearing on randomized routing and will thus be left out of this paper. Accordingly, the fault model allows multiple faults in a single column only, albeit the column can be any of the three columns. In the case of stuck switches in the middle column, it will be assumed that at least one middle switch is not stuck (for complete connectivity).

4 The Randomized Routing Algorithms for Clos networks

We will present two fault-tolerant randomized self-routing algorithms for Clos $C(p,q)$, namely, *the Single Randomization Algorithm*, and *the Multiple Randomization Algorithm*. In these algorithms, every input terminal processor will be assumed to have an independent uniform random number generator that generates integers in the range from 0 to $q-1$, corresponding to the local labels of the output ports in every switch of the left column.

The Single Randomization Algorithm

To route a permutation f, every input terminal s forms its control tag $z'x'y'$ to its destination $f(s) = [x'y']$ by selecting z' from $\{0,1,\ldots,q-1\}$ uniformly randomly and independently of other input terminals. Then, every input terminal s attempts to establish the corresponding path $s \rightarrow f(s)$ until it succeeds; when a path is established, its corresponding message is sent.

he Multiple Randomization Algorithm

The Multiple Randomization Algorithm differs from the Single Randomization Algorithm only in the repeated randomization. Specifically, after an input terminal i randomly selects z' and forms its control tag $z'x'y'$, if the path $i \rightarrow f(i)$ conflicts with other paths, the input terminal i does not commit itself to the same z' in the future cycles. Rather, every time an input terminal i experiences a conflict at the beginning of a cycle, it randomly selects a **new** z' and tries to establish the path $i \rightarrow f(i)$ in the next cycle. This is repeated until the path is established. Note that the repeated selection of z' does not incur additional overhead because it is done during the otherwise idle wait of input terminal i.

The conceptual rationale behind this multiple randomization method is that when a conflict occurs, some *clustering* of paths has occurred, so the repeated randomization helps to *uncluster* some of the paths and reduce the delay. *Reclustering* may take place, but the repeated randomization again takes care of the new clusters.

Fault Tolerance in the Randomized Algorithms

We now discuss how switch faults are tolerated in these two algorithms. If the faulty (i.e., stuck) switches are in the left column, the two routing algorithms remain the same except that no randomization takes place in the stuck switches. If the stuck switches are in the right column, it is assumed that the labels (i.e, locations) and settings of these switches are known to very input terminal. When a permutation f is to ،e routed, every input terminal s checks first if its destination is an output of a stuck switch. If so, the input terminal determines its unique control tag $z'x'y'$ where $f(s) = [x'y']$. Note that this control tag is unique because the path $s \rightarrow f(s)$ must be unique in this case. Note also that z' is easily computable because s knows the stuck setting of the right switches. If, on the other hand, the destination $f(s)$ is an output of a fault-free switch, the input terminal s carries out its original random selection unaffected.

If the stuck switches are in the middle column, a slightly different adjustment has to be made. Let F be the set of the labels of the stuck middle switches. When routing a permutation f, if a source $[xy]$ chooses its z' from F, then the path $[xy] \rightarrow f[xy]$ goes through the middle stuck switch $[z']_{middle}$ and is thus forced to go to a certain right switch. Formally, if $z' \in F$ (and $\alpha_{z'}$ is the stuck setting of switch $[z']_{middle}$), then the path $[xy] \rightarrow f[xy]$ is:

$$[xy]_{left} \rightarrow [xz']_{left} \longrightarrow [z'x]_{middle} \rightarrow$$
$$[z'\alpha_{z'}(x)]_{middle} \longrightarrow [\alpha_{z'}(x)z']_{right} \rightarrow f[xy]$$

implying that $f[xy]$ is an output of switch $[\alpha_{z'}(x)]_{right}$. Therefore, source $[xy]$ may select a z' that belong to F if and only if its destination is an output of the right switch $[\alpha_{\alpha_{z'}(x)}]_{right}$. Consequently, when routing f, every input terminal may choose its z' uniformly randomly from the set

$$A([xy],f) = \{z' \mid z' \notin F \text{ or } (z' \in F \text{ and } f([xy]) \text{ is an output of switch } [\alpha_{z'}(x)]_{right})\}.$$

This range of z' is easily determinable because the α's are known to each input terminal.

5 Performance Analysis

Probabilistic analysis of the communication delay of permutations under the Single Randomization Algorithm will be carried out in this section. The Multiple Randomization Algorithm seems much harder to analyze theoretically and will thus be studied by simulation only in the next section. It will prove convenient to analyze the delay in the Single Randomization Algorithm in the fault-free case first. The faulty case will follow later.

5.1 Analysis of the Fault-Free Single Randomization

Let f be an arbitrary, fixed permutation to be routed in $C(p,q)$ using the Single Randomization Algorithm. Every input terminal s of $C(p,q)$ will send a single message M_s to the output terminal $f(s)$ of $C(p,q)$. The maximum time delay of M_s to reach its destination will be probabilistically modeled and shown to follow a **binomial distribution**.

Throughout this paper, let

- f = the permutation to be routed.
- R = an arbitrary, fixed path $[at]_{left} \longrightarrow [ta]_{middle} \rightarrow [tb]_{middle} \longrightarrow [bt]_{right}$, from output port $[at]$ of the left column to the input port $[bt]$ of the right column (see Figure 2).
- X = the (random) number of source-destination paths that conflict with R when f is being randomly routed. The maximum time delay experienced by any message M that needs to use all of R is modeled by X.

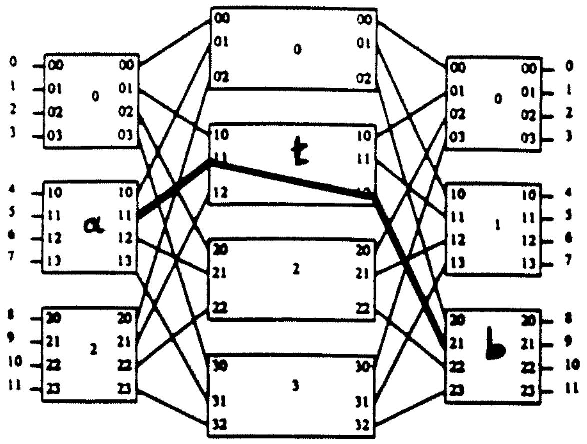


Figure 2: A Fixed Path R (in dark lines)

• $C = \{$source s | (s is an input of switch $[a]_{left}$) or ($f(s)$ is output of switch $[b]_{right}$)$\}$. In other terms, C is the set of the sources s such that the (random path) $s \rightarrow f(s)$ potentially conflicts with R. The potentiality may or may not become an actuality depending on which choices the randomized algorithm makes. Note that C depends on f, a and b, but not on the randomized algorithm.

• $c = |C|$. Note that $q \leq c \leq 2q$.

Theorem 1 *The random variable X follows the binomial distribution $B(c, \frac{1}{q})$.*

Proof: Let x_s be a Bernoulli random variable for every s in C, where $x_s = 1$ if the path $s \rightarrow f(s)$ selected by the algorithm conflicts with R; otherwise, $x_s = 0$. Thus, $X = \sum_{s \in C} x_s$. It can be seen that for every $s \in C$, the selected path $s \rightarrow f(s)$ conflicts with R if and only if it goes through the same middle switch as R, which is $[t]_{middle}$. Therefore, for every $s \in C$, the selected path $s \rightarrow f(s)$ conflicts with R if and only if the source terminal s selects the output port of local label t in the left column. As every source selects one out of q choices uniformly randomly, s chooses t with probability $\frac{1}{q}$. It follows that $Pr[x_s = 1] = \frac{1}{q}$. Thus, every x_s is a Bernoulli random variable with parameter $\frac{1}{q}$. In addition, since the sources make their random choices independently, the x_s's must be independent. Consequently, X follows the binomial distribution $B(c, \frac{1}{q})$. □

It follows from the previous theorem that the average value $E(X)$ is $\frac{c}{q}$ which is between 1 and 2 because $q \leq c \leq 2q$. More importantly, high probabilistic bounds on the value of the delay X will next be derived. We will first prove a useful theorem about binomial distributions. The reader may wish to skip the proof.

Theorem 2 *Let Y be a random variable that follows the binomial distribution $B(n,p)$. Let $\lambda = np = E(Y)$. Then, the following holds:*

(1) For every $k \geq 2\lambda$, $Pr[Y \leq k] \geq e^{-\lambda} \sum_{i=0}^{k} \frac{\lambda^i}{i!}$
(2) For all $\gamma \geq \lambda$ and for $k \geq 2\gamma$, $Pr[Y \leq k] \geq e^{-\gamma} \sum_{i=0}^{k} \frac{\gamma^i}{i!}$

Proof: (1) Let r be an integer $\geq 2\lambda + 1$. $Pr[Y = r] = \binom{n}{r} p^r (1-p)^{n-r}$ which can be shown to be equal to

$$Pr[Y = r] = \frac{\lambda^r}{r!}(1 - \frac{\lambda}{n})^n \prod_{l=0}^{r-1} \frac{n-l}{n-\lambda}$$

where $\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} = \frac{n}{n-\lambda} \times \frac{n-1}{n-\lambda} \times \ldots \times \frac{n-(r-1)}{n-\lambda}$. It will be shown next that $(1 - \frac{\lambda}{n})^n \leq e^{-\lambda}$ and $\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} \leq 1$.

Since $(1 - \frac{\lambda}{n})^n$ is an increasing function in n and converges to $e^{-\lambda}$, it follows that $(1 - \frac{\lambda}{n})^n \leq e^{-\lambda}$.

As for the claim $\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} \leq 1$, two cases will be distinguished based on whether r is even or odd. In each case, the r terms in the product will be regrouped as follows. For r even,

$$\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} = \prod_{i=0}^{\lfloor \frac{r-1}{2} \rfloor} [\frac{(n-i)[n-(r-1-i)]}{(n-\lambda)^2}],$$

and for r odd,

$$\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} = \frac{n-\frac{r-1}{2}}{n-\lambda} \prod_{i=0}^{\frac{r-1}{2}-1} [\frac{(n-i)[n-(r-1-i)]}{(n-\lambda)^2}].$$

The term $[\frac{(n-i)[n-(r-1-i)]}{(n-\lambda)^2}]$ can be shown to be an increasing function of i for $i \leq \frac{r-1}{2}$ (because the numerator is a parabola in i). Therefore, $[\frac{(n-i)[n-(r-1-i)]}{(n-\lambda)^2}] \leq [\frac{(n-\frac{r-1}{2})[n-(r-1-\frac{r-1}{2})]}{(n-\lambda)^2}] = [\frac{(n-\frac{r-1}{2})}{(n-\lambda)}]^2$ which is ≤ 1 when $r \geq 2\lambda + 1$. Thus, whether r is even or odd, $\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} \leq 1$.

As a result, we have $Pr[Y = r] \leq \frac{\lambda^r}{r!} e^{-\lambda}$. Hence, $Pr[Y \geq r] = \sum_{i=r}^{n} Pr[Y = i] \leq e^{-\lambda} \sum_{i=r}^{n} \frac{\lambda^i}{i!} \leq e^{-\lambda} \sum_{i \geq r} \frac{\lambda^i}{i!} = e^{-\lambda}(e^{\lambda} - \sum_{i=0}^{r-1} \frac{\lambda^i}{i!}) = 1 - e^{-\lambda} \sum_{i=0}^{r-1} \frac{\lambda^i}{i!}$. (Note that we made use of the well-known formula $e^{\lambda} = 1 + \frac{\lambda}{1!} + \frac{\lambda^2}{2!} + \ldots$).
Consequently, $Pr[Y \leq r-1] = 1 - Pr[Y \geq r] \geq e^{-\lambda} \sum_{i=0}^{r-1} \frac{\lambda^i}{i!}$ for $r \geq 2\lambda + 1$. Letting $k = r - 1$ which is $\geq 2\lambda$, part (1) of the theorem follows.

(2) It can be seen that $\frac{e^{-x} x^r}{r!}$ is an increasing function of x for $x \leq r$. Therefore, $Pr[Y = r] \leq \frac{\lambda^r}{r!} e^{-\lambda} \leq \frac{\gamma^r}{r!} e^{-\gamma}$ for $\lambda \leq \gamma \leq r$. Thus, $Pr[Y = r] \leq \frac{\gamma^r}{r!} e^{-\gamma}$ for $r \geq 2\gamma + 1$. The rest of the proof is the same as in the preceding paragraph. □

Since $X \sim B(c, \frac{1}{q})$, it follows that $\lambda = \frac{c}{q} \leq \frac{2q}{q} \leq 2$. Therefore, for $k \geq 4$, $Pr[X \leq k] \geq e^{-2} \sum_{i=0}^{k} \frac{2^i}{i!}$. By evaluating $e^{-2} \sum_{i=0}^{k} \frac{2^i}{i!}$ for $k = 4, 5, 6, 7, 8, 9$, we derive

$Pr[X \leq 4] \geq .94734$, $Pr[X \leq 5] \geq .98343$
$Pr[X \leq 6] \geq .99546$, $Pr[X \leq 7] \geq .99890$
$Pr[X \leq 8] \geq .99972$, $Pr[X \leq 9] \geq .99995$

These overwhelming probability figures, which hold for arbitrary f, p and q, clearly reveal the extreme communication speed of the Single Randomization Algorithm.

Since X is indicative of the delay of only a single (though arbitrary) message in a permutation, and since the probabilistic bounds are conservative lower bounds, simulation is needed to evaluate the entire delay of a permutation in actual networks. This will be carried in Section 6.

5.2 Analysis of Single Randomization in the Presence of Switch Faults

Let f, R and X denote the same entities as before, and assume there are some S stuck switches in one of the three columns. A source-destination path $s \rightarrow f(s)$ is said to be *a stuck path* if it goes through a stuck switch. Recall that X is the number of selected source-destination paths that conflict with R. Let Y be the number of stuck paths that conflict with R, and Z the number of non-stuck paths that conflict with R. Clearly, $X = Y + Z$. By studying Y and Z we will be able to derive probabilistic estimates of X. This will be carried out in three cases depending on which column has the stuck switches.

CASE 1: The Stuck Switches in the Left Column

The distribution of Z is addressed first. Observe that for every source s of a stuck switch, there is a unique path from s to $f(s)$, that is, the stuck paths are unique and non-random. Let n be the number of sources s that are input of non-stuck left switches such that the random path $s \rightarrow f(s)$ potentially conflicts with R. Following the same reasoning as in the fault-free case, it is concluded that $Z \sim B(n, \frac{1}{q})$. Noting that $n + Y = c$, where c is as defined in the previous subsection, it follows that $n \leq 2q$ because $c \leq 2q$.

The value of Y is treated next. The stuck paths that conflict with R must go through the middle switch $[t]_{middle}$ of R. In particular, their sources must be inputs s of stuck left switches $[x]_{left}$ such that $s = [x\alpha_x^{-1}(t)]$. In other terms, out of every stuck switch there can originate at most one stuck path that conflicts with R. Therefore, $Y \leq$ the number of faulty switches.

Since $X = Y + Z$ and $Z \sim B(n, \frac{1}{q})$ and $E(Z) = \frac{n}{q} \leq 2$, we derive from Theorem 2 that for all $k \geq 4$,

$$Pr[X \leq Y + k] \geq Pr[X \leq k] \geq e^{-2} \sum_{i=0}^{k} \frac{2^i}{i!}.$$

In particular, we have

$$\begin{array}{ll} Pr[X \leq Y+4] \geq .94734, & Pr[X \leq Y+5] \geq .98343 \\ Pr[X \leq Y+6] \geq .99546, & Pr[X \leq Y+7] \geq .99890 \\ Pr[X \leq Y+8] \geq .99972, & Pr[X \leq Y+9] \geq .99995 \end{array}$$

These figures show that the smaller the number of stuck switches is, the smaller Y is and the better these probabilistic bounds are. In some sense, Y represents the amount of delay degradation due to faulty switches.

Clearly, Y is equal in the worst case to the number of stuck switches. However, to get a more realistic value for Y, Y will be averaged over all its possible values corresponding to all the $N!$ permutations, assuming that every permutation is equally likely.

$Pr[Y = i]$=[*# of permutations that map exactly i stuck sources to some i outputs of switch* $[b]_{right}$]$/N!$

$$Pr[Y = i] = \frac{\binom{S}{i}\frac{q!}{(q-i)!}\frac{(N-q)!}{(N-q-(S-i))!}(N-S)!}{N!}$$

$$Pr[Y = i] = \frac{\binom{N-S}{q-i}\binom{S}{i}}{\binom{N}{q}}.$$

Therefore, Y follows a **hypergeometric distribution**. It can now be concluded that the expected value of Y is $\mu_Y = \frac{Sq}{N} = \frac{S}{p} \leq 1$ and the variance $VAR(Y) = \frac{S(S-1)q(q-1)}{N(N-1)} + \frac{Sq}{N} - \frac{S^2q^2}{N^2}$. Since $\mu_Y \leq 1$, it follows that he entire delay of an average permutation in the presence of stuck switches in the left column is degraded by at most one cycle. To get tighter estimates of Y, note that $VAR(Y)$ can be easily shown to be < 1, and therefore, the standard deviation $\sigma_Y < 1$. Applying the well-known Chebychev formula:

$$Pr[\mu_Y - k\sigma_Y < Y < \mu_Y + k\sigma_Y] \geq 1 - \frac{1}{k^2},$$

we conclude that

$$Pr[Y \leq k] \geq 1 - \frac{1}{k^2}.$$

Consequently, $Pr[Y \leq 3] \geq 0.88$, $Pr[Y \leq 4] \geq 0.93$ and $Pr[Y \leq 5] \geq 0.96$. In particular, the delay degradation Y is at most 3 cycles for at least 88% of the $N!$ permutations. This shows that for the overwhelming majority of permutations, the performance degradation due to switch faults is very small indeed.

CASE 2: The Stuck Switches in the Right Column

Due to the symmetry between the left and right column in Clos networks, the delay $X = Y + Z$ can be similarly shown to satisfy: $\mu_Y \leq 1$, $\sigma_Y < 1$ and the distribution of Z is binomial of mean ≤ 2. This leads to the same conclusions as in the previous case.

CASE 3: The Stuck Switches in the Middle Column

We will first treat the case where the path R does not go through a stuck switch. The other case is handled later.

Observe that the source-destination paths that conflict with R must go through the same middle switch as R (i.e., switch $[t]_{middle}$). Therefore, if switch $[t]_{middle}$ is not stuck, then the source-destination paths $s \rightarrow f(s)$ that conflict with R are non-stuck paths. Therefore, $Y = 0$ and $X = Z$. Let C be as previously defined, that is, the set of sources s such that $s \rightarrow f(s)$ potentially conflicts with R. Let also x_s, for every source $s \in C$, be a Bernoulli variable such that $x_s = 1$ if the randomly chosen path $s \rightarrow f(s)$ conflicts with R, $x_s = 0$ otherwise. Recall that $X = \sum_{s \in C} x_s$. Recall also that the range of z' for every source s is $A(s, f)$ as defined in Section 4 for the case where the stuck switches are in the middle column. Let $q_s = |A(s, f)|$. It can be argued as in Subsection 5.1 that every x_s is a Bernoulli variable of parameter $\frac{1}{q_s}$. The value of q_s depends on the number and locations of the faulty middle switches, on the permutation f, and on s itself. Therefore, X is the sum of independent but differently distributed Bernoulli random variables. Thus, X does not necessarily follow a binomial distribution. This makes the probabilistic analysis of X harder. However, we will find next some useful bounds on the mean (μ_X) of X.

Assume that the number of stuck switches is a fraction εq of the number of the middle switches, $0 \le \varepsilon < 1$. It is clear from the definition of $A(s,f)$ that the size q_s of the range of z' for every source s is at least $q - \varepsilon q = (1-\varepsilon)q$. We then have

$$\mu_X = \sum_{s \in C} \mu_{x_s} = \sum_{s \in C} \frac{1}{q_s} \le \frac{c}{(1-\varepsilon)q}.$$

Since $c \le 2q$, it follows that

$$\mu_X \le \frac{2}{1-\varepsilon}.$$

Therefore, the smaller the number of stuck switches, the smaller the bound on μ_X. Equivalently, the larger the the number of stuck switches, the more likely that the expected value of X is high. This notion is exhibited more precisely in our simulation results presented in the next section.

We now treat the case when the middle switch $[t]_{middle}$ of R is stuck. Every source-destination path that conflicts with R must go through the switch $[t]_{middle}$. Because switch $[t]_{middle}$ is stuck, every source-destination path that conflicts with R must conflict over ALL the links of R. Therefore, the source-destination paths that conflict with R have their sources in the same left switch $[a]_{left}$ as R and their destinations in the same right switch $[b]_{right}$ as R. Let $C' = \{\text{source } s \mid (s \text{ is an input of switch } [a]_{left}) \text{ and } (f(s) \text{ is an output of switch } [b]_{right})\}$, and $c' = |C'|$. Clearly, $c' \le q$.

By carrying out the same analysis as in the previous two paragraphs, we get $\mu_X \le \frac{c'}{(1-\varepsilon)q} \le \frac{1}{1-\varepsilon}$, leading to the same conclusion that the mean of X increases with ε.

In summary, this section has presented theoretical analysis of the fault-tolerant Single Randomization Algorithm. In fault-free Clos, it was found that the delay is 4–8 cycles with an overwhelming probability. In faulty Clos, it was found that the daley is hardly sensitive to faulty switches in the left or right column, but it is sensitive to the number of faulty switches in the middle column.

Despite the reasonably tight probabilistic estimates of the delay, simulations are still needed for several reasons. First, the conservative nature of the probabilistic lower bounds on the delay and the difficulty of precisely measuring the delay degradation when the faulty switches are in the middle column make simulations the only means of measuring the actual delay. Second, as was indicated early on, the Multiple Randomization Algorithm seems hard to analyze theoretically, making the simulation a proper alternative method of mesuring its performance and comparing it to the Single Randomization Algorithm.

6 Simulation

We have simulated the two algorithms on Clos networks $C(p,q)$ for $4 \le q \le 32$, taking $p = q$ for convenience. For each algorithm, for each value of q, and for each randomly selected set of stuck switches (with randomly selected stuck settings) in one of the three columns, we did the following:

1) Several hundred randomly selected permutations were routed.

2) For each permutation we recorded the total number of cycles needed to complete routing the N messages of the permutation.

3) We averaged the number of cycles per permutation over the several hundred routed permutations.

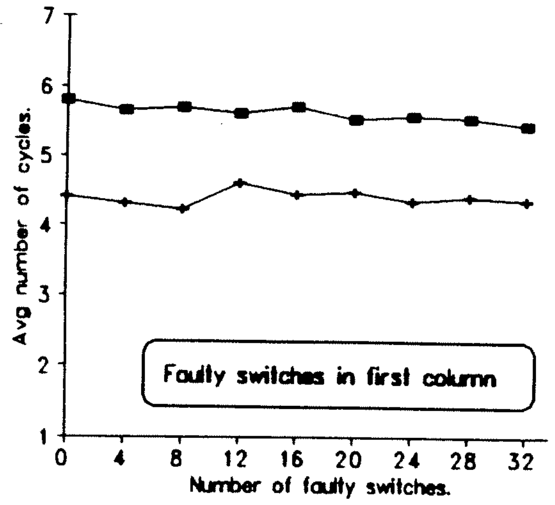
The results of the above simulation study are summarized in Figure 3. The first three plots show the effects of the number of switch faults on the routing delay in Clos $C(32,32)$. The last three plots show the effect of the switch size on the routing delay assuming a single faulty switch.

In Figure 3-(a,b,c), the average number of cycles required to route a permutation is plotted as a function of the number of faulty switches in the left column, right column, and middle column, respectively. Two major observations can be made. First, the 3 plots show that the Multiple Randomization Algorithm yields better permutation delays than the Single Randomization Algorithm. The delay under multiple randomization is consistently shorter by roughly one cycle than the delay under single randomization. Second, plots (a) and (b) show that the delay of permutations is hardly affected by the number of faulty switches in the left or right columns, confirming what was asserted theoretically. Plot (c) clearly shows that the delay of permutations under both algorithms increases as the number of stuck middle switches increases. This confirms the approximate behavior observed in the theoretical analysis. It should be pointed out that the delay increase is about one cycle when the number of faulty middle switches is 20, and about 1.5 cycles (in multiple randomization) and 3 cycles (in single randomization) when all the middle switches are faulty. We note that when all the middle switches are stuck, we made sure that the stuck settings kept every destination accessible from every source.

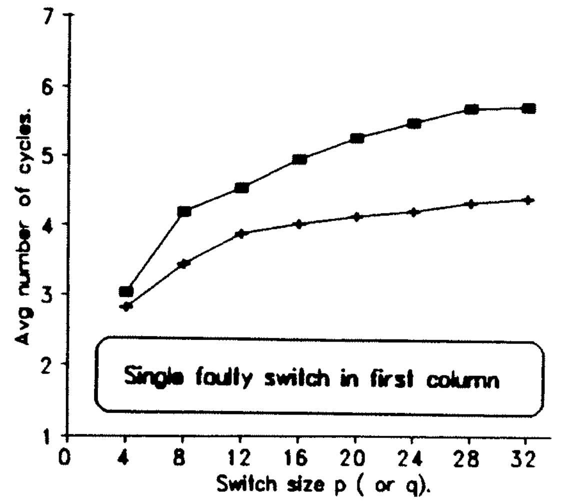
This observed behavior clearly shows the great fault tolerance capabilities of randomized routing, especially when using multiple randomization. The short delays of 4.5–6 cycles per permutation, with or without faults, makes the Multiple Randomization Algorithm very attractive for routing, and that in turn makes Clos networks highly desirable networks for parallel computer systems of up to 1024 processors.

In Figure 3-(d,e,f), we plot the number of cycles per permutation as a function of the switch size, assuming that there is a single faulty switch in the left, middle, and right column, respectively. The intent is to examine the effect of the switch size on the performance of the two routing algorithms. Again it can be observed that the Multiple Randomization Algorithm consistently gives better performance than the Single Randomization Algorithm, and that the former algorithm tends to yield a stationary delay of 4 cycles (!) for switch sizes ≥ 16 . The delays are smaller for smaller switch sizes.

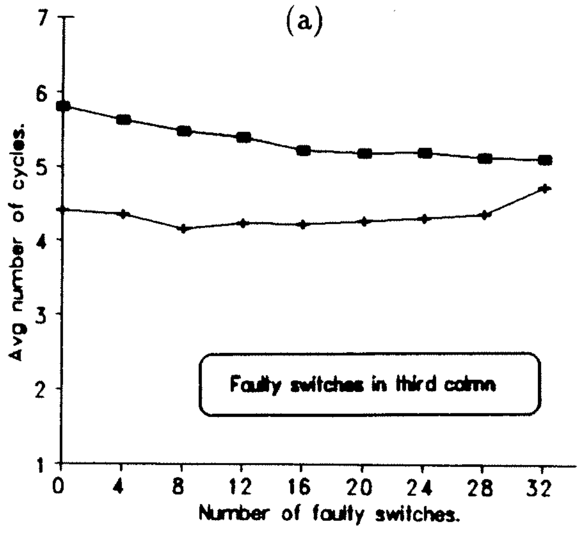
In summary, the simulation results are certainly consistent with the probabilistic analysis. They further reinforce the conclusion that the proposed randomized routing algorithms deliver high performance even in the presence of one or more faulty stuck-at switches in any one column, and that multiple randomization is better than single randomization.



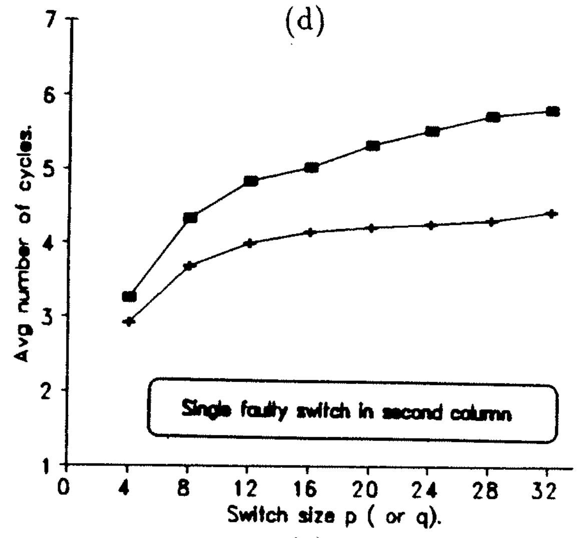
(a)



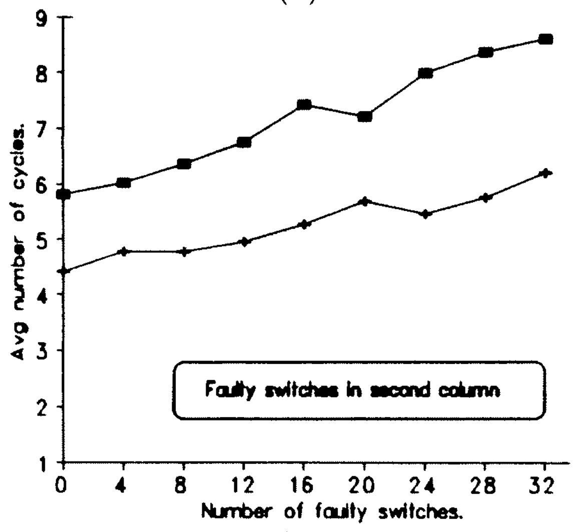
(d)



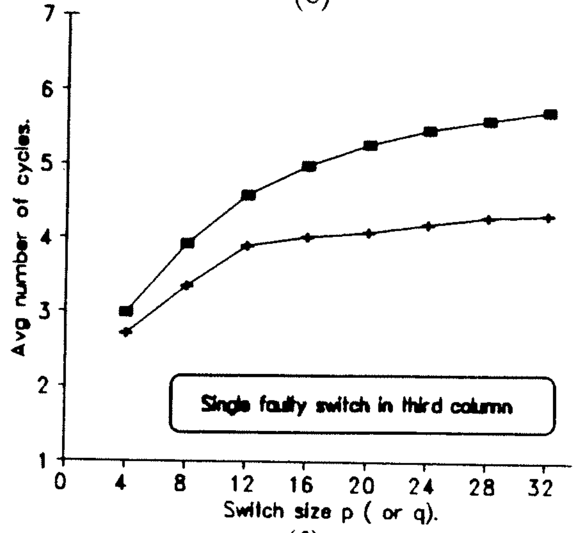
(b)



(e)



(c)



(f)

—■— Single Randomization —✦—Multiple Randomization

Figure 3: Simulation Results

7 Conclusion and Future Directions

This paper introduced and studied two randomized self-routing algorithms for Clos networks. The performance analysis and the simulations have shown that the communication delay of any permutation is very small, even in the presence of multiple switch faults in any single column. Multiple randomization was shown to yield shorter delays than single randomization. The delay under multiple randomization is below 6.5 cycles per permutation in Clos networks of up to 1024 processors, even in the presence of many faulty switches.

The ease of implementation of our routing algorithms, their routing speed, their universality, and their very low communication delay even in the presence of faults make these algorithms superior to any other known universal routing algorithm for Clos. They also make Clos networks superior to the non-universal banyan MIN's. These advantages and the comparative VLSI cost make Clos networks highly practical and attractive for large parallel systems.

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, N0. 5, pp. 443–454, May 1982.

[2] G. B. Adam, D. P. Agrawal and H. J. Siegel, "A Survery 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. Chen and K.G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers," *IEEE Trans. Comput.*, Vol. 39, N0. 12, pp. 1406–1416, Dec. 1990.

[7] 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.

[8] C. Clos, "A study of non-blocking switching networks," *Bell System Tech. J.* Vol. 32, pp. 406–424, 1953.

[9] 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.

[10] 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.

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

[12] 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.

[13] D. Mitra and R. A. Cieslak, "Randomized Parallel Communications on an Extension of the Omega Network," *J. ACM*, Vol. 34, No. 4, pp. 802–824, Oct. 1987.

[14] 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.

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

[16] J. P. Shen and J. P. Hayes, "Fault-Tolerance of Dynamic Full–Access Interconnection Networks," *IEEE Trans. Comput.*, March 1984, pp. 241–248.

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

[18] 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, 1986.

[19] L. G. Valiant, "A Scheme for Fast Parallel Communication," *SIAM J. Comput.*, Vol. 11, No. 2, pp. 350–361, May 1982.

[20] 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.

[21] C. L. Wu and T. Y. Feng, "Fault Diagnosis for a Class of Multistage Shuffle/Exchange Networks," *IEEE Trans. on Comput.*, Vol. C-30, pp. 743–758, Oct. 1981.

[22] Y.-M. Yeh and T.-Y. Feng, "Fault-Tolerant Routing on a Class of Rearrangeable Networks," *International Conference on Parallel Processing*, Vol. I, pp. 305–312, 1991.

[23] A. Youssef and B. Arden, "A new approach to fast control of $r^2 \times r^2$ 3-stage Benes networks of $r \times r$ crossbar switches," *Proc. 17th Int'l Symp. Comp. Arch.*, Seattle, pp. 50–59, May 1990.