Computers Elect. Engng Vol. 19, No. 6, pp. 419–429, 1993
Printed in Great Britain.

0045-7906/93 $6.00 + 0.00

RANDOMIZED SELF-ROUTING ALGORITHMS FOR CLOS NETWORKS

ABDOU YOUSSEF
Department of EECS, The George Washington University, Washington, DC 20052, U.S.A.

(Accepted in final revised form 6 January 1993)

Abstract—As the VLSI technology makes large crossbar switches affordable, Clos networks have become a feasible option of large interconnection networks. However, to make these networks practical and useful, efficient routing algorithms need to be developed. This paper will develop and study several randomized routing algorithms for Clos networks. The algorithms are based on the idea that if the first column of Clos is set to some configuration somehow, then the resulting network becomes self-routed using the destination addresses. Each of the randomized algorithms sets the first column to a configuration selected by a random process. The algorithms are then self-routed and take no computation time to set the switches. Probabilistic analysis and simulation measurements of the communication delay of permutation routing are conducted. It is shown that the communication delay of any permutation is 3–6 cycles in networks of up to 1024 processors. Although other routing algorithms route arbitrary permutations in one cycle over Clos/Benes networks and 2 cycles over δ networks, these algorithms take prohibitively large times to compute the appropriate switch settings, while our randomized algorithms are self-routed and spend NO time on computing the switch settings. This makes our algorithms superior to any universal nonrandomized routing algorithm for Clos/Benes networks or δ networks. The speed, universality and ease of implementation of our randomized algorithms make Clos networks highly attractive for large parallel computer systems.

Key Words: Clos networks, communication delay, performance analysis, permutations, randomized self-routing.

1. INTRODUCTION

Clos networks [1] and Benes networks [2] are the best known **universal** multistage interconnection networks. These two networks enjoy several advantages over banyan multistage interconnection networks [3] (e.g. the Omega network [4], the indirect binary n-cube [5], and the baseline network [6]) and graph interconnection networks (e.g. meshes and hypercubes). They are superior to banyan multistage interconnection networks in universality (i.e. ability to route all permutations) as well as potential for fault tolerance due to the multiplicity of paths between source–destination pairs. They are also superior to graph interconnection networks in two respects. First, they require their processors to have only one input port and one output port. This requirement is less than what is needed in most graph interconnection networks except linear arrays and rings which are of limited use. Second, with efficient Clos/Benes routing algorithms, the problem of task-to-processor assignment is straightforward in Clos and Benes but is very costly in graph interconnection networks if low communication overhead is to be achieved.

Up against these advantages Clos networks and, to a lesser extent, Benes networks have had (until recently) the disadvantage of higher hardware cost and slower routing speed. If the hardware cost is measured by the number of 2×2 switching elements as was the case before the advent of VLSI, a Clos network for an N-processor system costs $O(N\sqrt{N})$ switching elements, while all the other MINs cost $O(N \log N)$. As for routing, there are two general Clos routing algorithms, the first by Opferman and Taso-Wu [7] and the second by Lev *et al.* [8]. The first takes $O(N^2)$ time to determine the appropriate switch settings to realize a given permutation. The second algorithm takes $O(N \log^2 N)$ time and can be reduced to $O(N \log N)$ if N is a power of 2. These algorithms are very costly, even prohibitive for on-line routing, and certainly much slower than the self-routing algorithms of banyan MINs. The situation has been partially improved by a fast self-routing approach introduced lately by Youssef and Arden [9]. The approach self-routes many interesting classes of permutations such as the permutations required by FFT, bitonic sorting, multigrid computations and the permutations realizable by the Omega network, but the approach does not apply to arbitrary permutations and has no fault-tolerance potential. The situation with Benes

routing is similar. Fast Benes routing algorithms have been found for special classes of permutations such as Lenfant's frequently used permutations [10], Nassimi and Sahni's bit-permute-complement permutations [11], and Raghavandra and Boppana's linear-complement permutations [12]. But for arbitrary permutations, Benes routing takes a costly $O(N \log N)$ sequential time [13]. Although two parallel routing algorithms for Benes have been developed where one takes $O(N)$ time [13] and the other $O(\log^2 N)$ time [14], they are not practical because the first is still too costly and the second requires an unrealistic fully-connected (i.e. complete) network of N nodes to run.

However, the recent advances of VLSI eliminate or greatly reduce the cost disadvantage of Clos networks. With VLSI, the number of 2×2 switching elements has become a largely irrelevant metric. Rather, the VLSI area is now the true measure of cost. By this measure, the costs of Clos networks are comparable to those of other MINs. In other words, large crossbar switches can now be built, making the synthesis of Clos networks affordable. IBM GF11 [15] is an example of a Clos-interconnected machine which utilizes 24×24 switches.

This paper will largely eliminate the routing disadvantage of Clos networks by proposing randomized self-routing algorithms and showing that they enable Clos networks to deliver a very high routing performance. These algorithms are universal, that is, they route all permutations. They are also self-routing (i.e. distributed) and, therefore, do not incur any computation time to determine the switch settings. They are based on the idea that when routing a permutation on a Clos network (which has 3 columns of switches), if the setting of the first column is determined somehow, then the setting of the remaining two columns can be found locally by the switches themselves using the destination addresses. In these routing algorithms, the setting of the first column is determined **randomly**.

The performance of the randomized algorithms will be studied using probabilistic analysis and simulation. Our probabilistic analysis will show that the algorithms route any permutation in 4–8 network cycles with overwhelmingly high probability. Note that a network cycle is the time period needed to: (1) establish nonconflicting source–destination paths; and (2) deliver the corresponding messages. The simulations confirm the theoretical results and exhibit even better performance in practice: 3–6 cycles/permutation. Furthermore, the best algorithm among the ones developed will be shown to deliver an almost deterministic delay of 4 cycles.

The paper is organized as follows. The next section will given an overview of Clos networks. Section 3 will present three randomized routing algorithms. Section 4 will conduct probabilistic analysis of the communication delay incurred in the algorithms. Section 5 will report the simulation findings of the communication delay. Concluding remarks and future directions will be presented in Section 6.

2. OVERVIEW OF CLOS NETWORKS

Let p and q be two positive integers and $N = pq$ throughout. A Clos network $C(p, q)$ 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 $0, 1, \ldots, p-1$, and the switches in the middle column are labeled $0, 1, \ldots, q-1$. The input ports and the output ports of every $n \times n$ switch have *local labels* $0, 1, \ldots, n-1$ ($n = p$ or p). 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 $[x]_{\text{left}}$, $[x]_{\text{middle}}$ or $[x]_{\text{right}}$, respectively. Similarly, every input/output port $[xy]$ of the left, middle or right column will be denoted $[xy]_{\text{left}}$, $[xy]_{\text{middle}}$, or $[xy]_{\text{right}}$. The interconnection between the first two columns links every output port $[xy]_{\text{left}}$ to input port $[yx]_{\text{middle}}$. Similarly, the interconnection between the middle column and the right column links output port $[yx]_{\text{middle}}$ to input port $[xy]_{\text{right}}$. These interconnections will be referred to as the first and second interconnections, respectively. Figure 1 shows a Clos network.

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 $[xy]_{\text{left}}$ of $C(p, q)$ to the output terminal $[x'y']$. The message first enters the input port $[yx]_{\text{middle}}$, then, using

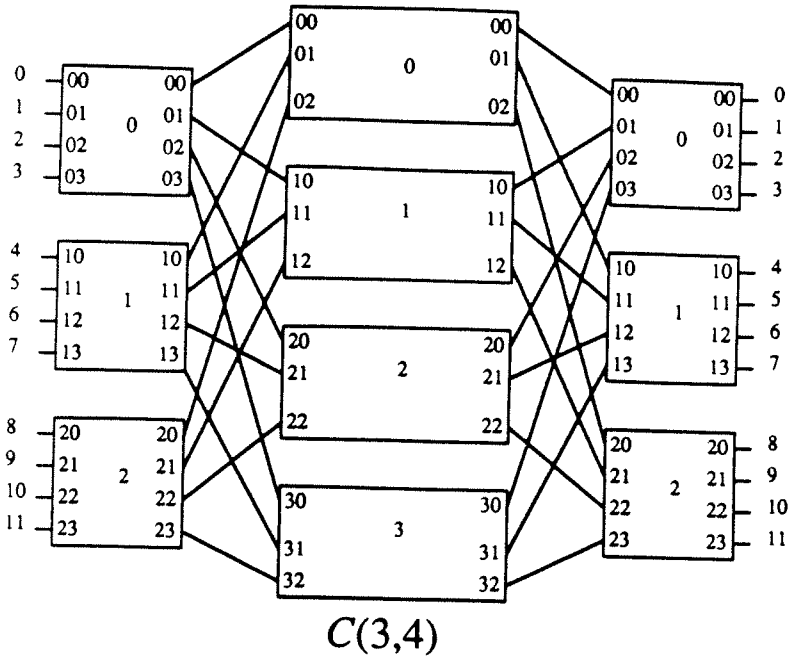

Fig. 1. A Clos network.

the digit x' of the destination address $[x'y']$, the message exits through output port $[yx']_{\text{middle}}$. Afterwards, it enters the input port $[x'y]_{\text{right}}$ and then, using digit y' of the destination address $[x'y']$, it reaches output port $[x'y']_{\text{right}}$, 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' (output of switch $[v]$ in the left column), $0 \leqslant z' \leqslant q - 1$, 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 link input port $[vw]$ to output port $[vz']$ of the first column, and from there establish the path to the input terminal $[x'y']$ as explained above. The way z' is selected characterizes the routing algorithm. In this paper, z' will be selected randomly in three different ways corresponding to the three randomized algorithms that will be explained later.

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 other paths are abolished awaiting future cycles. (Recall that cycle is the time period needed to estabish nonconflicting source–destination paths and deliver the corresponding messages.) This method clearly prevents deadlock and starvation when routing permutations.

Finally, it should be noted that because the communication delay of permutations is nondeterministic in randomized routing, some synchronization mechanism is needed to detect the completion of one permutation so that a new permutation can start. One possibe mechanism is to employ a single line bus linked to all the output terminals. Whenever a permutation is started, every output terminal puts "1" on the bus until it receives its message at which time it stops putting "1". As long as there is still one output terminal that has not received a message, there is "1" on the bus, that is, the bus is "hot". When all the output terminals have received their data, the bus becomes "cold", which is taken as a signal to all the processors that they can start their next permutation. Other synchronization mechanisms may also be used, but this is outside the scope of this paper.

3. THE RANDOMIZED ROUTING ALGORITHMS FOR CLOS NETWORKS

We will present three randomized routing algorithms for $C(p, q)$, namely, the *Single Randomization Algorithm*, the *Switch 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 labels of the output ports in every switch in the left column.

3.1. 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.

3.2. The Switch Randomization Algorithm

In the Single Randomization Algorithm conflicts occur in both the left and middle columns. If, however, the switches in the left column are set to one-to-one settings, conflicts in the left column will be eliminated and, consequently, the routing delay of permutations is likely to be reduced. This is accomplished in *the Switch Randomization Algorithm* described next.

Every switch in the left column is set to a one-to-one setting selected uniformly randomly from a set S of settings. Each switch is set independently of the other switches in the left column. After the left column is set, the network becomes a self-routed network that uses the destination addresses as control tags to control the middle and right columns.

The size of the set S has a bearing on the computational cost of the algorithm. The number of one-to-one settings of a $q \times q$ switch is $q!$. The set S from which to randomly select a setting for each left switch is in principal the set of all the $q!$ settings. But for reasonably large q ($q \geq 10$), the random selection from among $q!$ settings takes $O(q)$ time. Our probabilistic analysis conducted in Subsection 4.2 will show that the probability distribution of the communication delay is virtually the same whether S consists of all the $q!$ settings or just the q *cyclic shift* settings $\{\pi_j \mid 0 \leq j \leq q-1\}$, where $\pi_j(x) = x + j \bmod q$ for every $x = 0, 1, \ldots, q-1$. To select a cyclic shift setting uniformly randomly, it suffices to select j uniformly randomly from the set $\{0, 1, \ldots, q-1\}$ and set the switch to π_j. [By setting a switch to π_j means that every input port of local label x is linked to the output port of local label $\pi_j(x)$, $x = 0, 1, \ldots, q-1$.] This selection of j takes constant time (using a standard pseudo-random number generator). Therefore, we take S to be the set of these q cyclic shift settings.

It should be noted that the random selection of j for every left switch is carried out by some designated input terminal (i.e. procesor) directly linked to that switch. The selected j is then forwarded to the switch control to set the switch.

3.3. The Multiple Randomization Algorithm

Another way to improve the Single Randomization Algorithm is to employ multiple randomization in the left column.

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

The conceptual rationale behind this multiple randomization method is that when a conflict occurs, some *clustering* of paths occurred, so the repeated randomization opens the way 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.

4. THEORETICAL PERFORMANCE ANALYSIS OF THE ALGORITHM

In this section, probabilistic analysis of the communication delay in the first two routing algorithms will be carried out. As for the Multiple Randomization Algorithm, the theoretical analysis seems to be very complex. We will therefore analyze the performance of the latter algorithm via simulation only.

4.1. Performance analysis of 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 probabistically modeled and shown to follow a binomial distribution.

Let:

- f = the permutation to be routed.
- R = an arbitrary, fixed path $[at]_{\text{left}} \rightarrow [ta]_{\text{middle}} \rightarrow [tb]_{\text{middle}} \rightarrow [bt]_{\text{right}}$, from output port $[at]$ of the left column to the input port $[bt]$ of the right column.
- 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.
- $C = \{\text{source } s \mid (s \text{ is an input of switch } [a]_{\text{left}} \textbf{ or } (f(s) \text{ is output of switch } [b]_{\text{right}})\}$. 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 $q \leqslant c \leqslant 2q$.

Theorem 4.1. The random variable X follows the binomial distribution $B(c, 1/q)$.

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 $[t]_{\text{middle}}$. 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 $1/q$. It follows that $\Pr[x_s = 1] = 1/q$. Thus, every x_s is a Bernoulli random variable with parameter $1/q$. In addition, since the sources make their random choices independently, the x_ss must be independent. Consequently, X follows the binomial distribution $B(c, 1/q)$. ■

It follows from the previous theorem that the average value $E(X)$ is c/q which is between 1 and 2 because $q \leqslant c \leqslant 2q$. More importantly, high probabilistic bounds on the range of the maximum delay X will next be derived. We will first prove a useful theorem about binomial distributions.

Theorem 4.2. 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 $k \geqslant 2\lambda$, $\Pr[Y \leqslant k] \geqslant e^{-\lambda} \sum_{i=0}^{k} \frac{\lambda^i}{i!}$;

(2) for all $\gamma \geqslant \lambda$ and for $k \geqslant 2\gamma$, $\Pr[Y \leqslant k] \geqslant e^{-\gamma} \sum_{i=0}^{k} \frac{\gamma^i}{i!}$.

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

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

where

$$\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} = \frac{n}{n-\lambda} \times \frac{n-1}{n-\lambda} \times \cdots \times \frac{n-(r-1)}{n-\lambda}.$$

It will be shown next that

$$\left(1 - \frac{\lambda}{n}\right)^n \leqslant e^{-\lambda}$$

and

$$\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} \leqslant 1.$$

To this effect, assume that λ is constant and view $(1-\lambda/n)^n$ as a function of n. Let

$$g(n) = \ln\left(1-\frac{\lambda}{n}\right)^n = n \ln\left(1-\frac{\lambda}{n}\right).$$

The first derivative

$$g'(n) = \ln\left(1-\frac{\lambda}{n}\right) + \frac{\lambda}{n-\lambda}.$$

The second derivative

$$g''(n) = \frac{-\lambda^2}{(n-\lambda)^2} < 0.$$

Therefore, g' is a decreasing function of n for $n > \lambda$. Since $g'(n)$ goes to 0 as n approaches infinity, it follows that $g'(n) > 0$ for all $n > \lambda$. Thus, $g(n)$ is an increasing function in n. Consequently, $(1-\lambda/n)^n$ is an increasing function of n which is known also to converge to $e^{-\lambda}$. Hence

$$\left(1-\frac{\lambda}{n}\right)^n \leqslant e^{-\lambda}.$$

As for the claim

$$\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} \leqslant 1,$$

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:

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

and for r odd

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

The term $[(n-i)[n-(r-1-i)]/(n-\lambda)^2]$ can be shown to be an increasing function of i for $i \leqslant (r-1)/2$ (because the numerator is a parabola in i). Therefore;

$$\left[\frac{(n-i)[n-(r-1-i)]}{(n-\lambda)^2}\right] \leqslant \left[\frac{\left(n-\dfrac{r-1}{2}\right)\left[n-\left(r-1-\dfrac{r-1}{2}\right)\right]}{(n-\lambda)^2}\right] = \left[\frac{\left(n-\dfrac{r-1}{2}\right)}{(n-\lambda)}\right]^2$$

which is $\leqslant 1$ when $r \geqslant 2\lambda + 1$. Thus, whether r is even or odd:

$$\prod_{l=0}^{r-1} \frac{n-l}{n-\lambda} \leqslant 1.$$

As a result, we have $\Pr[Y = r] \leqslant (\lambda^r/r!)e^{-\lambda}$. It follows that:

$$\Pr[Y \geqslant r] = \sum_{i=r}^{n} \Pr[Y = i] \leqslant e^{-\lambda} \sum_{i=r}^{n} \frac{\lambda^i}{i!} \leqslant e^{-\lambda} \sum_{i \geqslant r} \frac{\lambda^i}{i!} = e^{-\lambda}\left(e^{\lambda} - \sum_{i=0}^{r-1} \frac{\lambda^i}{i!}\right) = 1 - e^{-\lambda} \sum_{i=0}^{r-1} \frac{\lambda^i}{i!}.$$

(Note that we made use of the well-known formula $e^{\lambda} = 1 + \lambda/1! + \lambda^2/2! + \ldots$). Consequently,

$$\Pr[Y \leqslant r-1] = 1 - \Pr[Y \geqslant r] \geqslant e^{-\lambda} \sum_{i=0}^{r-1} \frac{\lambda^i}{i!}$$

for $r \geqslant 2\lambda + 1$. Letting $k = r - 1$ which is $\geqslant 2\lambda$, part (1) of the theorem follows.

(2) It can be seen that $e^{-x}x^r/r!$ is an increasing function of x for $x \leqslant r$. Therefore

$$\Pr[Y = r] \leqslant \frac{\lambda^r}{r!} e^{-\lambda} \leqslant \frac{\gamma^r}{r!} e^{-\gamma} \quad \text{for} \quad \lambda \leqslant \gamma \leqslant r.$$

Thus,

$$\Pr[Y = r] \leqslant \frac{\gamma^r}{r!} e^{-\gamma} \quad \text{for} \quad r \geqslant 2\gamma + 1.$$

The rest of the proof is the same as in the preceding paragraph. ■

Since X is binomial $B(c, 1/q)$, it follows that $\lambda = c/q \leqslant 2q/q \leqslant 2$. Therefore, for $k \geqslant 4$

$$\Pr[X \leqslant k] \geqslant e^{-2} \sum_{i=0}^{k} \frac{2^i}{i!}.$$

By evaluating

$$e^{-2} \sum_{i=0}^{k} 2^i/i! \quad \text{for} \quad k = 4, 5, 6, 7, 8, 9,$$

we derive:

$$\Pr[X \leqslant 4] \geqslant 0.94734, \quad \Pr[X \leqslant 5] \geqslant 0.98343, \quad \Pr[X \leqslant 6] \geqslant 0.99546,$$

$$\Pr[X \leqslant 7] \geqslant 0.99890, \quad \Pr[X \leqslant 8] \geqslant 0.99972, \quad \Pr[X \leqslant 9] \geqslant 0.99995.$$

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 5.

4.2. Performance Analysis of Switch Randomization

Let f, R, a, b, t and X denote the same entities as before. Let C also denote the set of sources s such that the path $s \rightarrow f(s)$ potentially conflicts with R. Under the Switch Randomization Algorithm, conflicts occur over links in the second but not the first interconnection. Thus, a path potentially conflicts with R if and only if the destination of the path is an output port of the right switch $[b]_{\text{right}}$. Thus $c = |C| = q$. Furthermore, for every $s \in C$, the selected path $s \rightarrow f(s)$ conflicts with R if and only if that path goes through the same middle switch $[t]_{\text{middle}}$ as R, which is the case if and only if the selected one-to-one setting of the left switch of s links s to the output port of local label t. As the switch settings are one-to-one, from every left switch there can be at most one input terminal that conflicts with R no matter what the random setting is. Let then x_k, $0 \leqslant k \leqslant p - 1$, be a Bernoulli random variable such that $x_k = 1$ if the randomly selected setting of left switch $[k]_{\text{left}}$ causes one input of that switch to conflict with R, otherwise $x_k = 0$. Clearly,

$$X = \sum_{k=0}^{p-1} x_k.$$

As the switches are set independently, these x_ks are independent.

$\Pr[x_k = 1]$ will be evaluated next. Let C_k be the set of source input terminals of left switch $[k]_{\text{left}}$ such that their destinations are outputs of switch $[b]_{\text{right}}$ and let $c_k = |C_k|$. That is, C_k is the set of inputs of left switch $[k]_{\text{left}}$ that potentially conflicts with R. Clearly, C_k depends only on f, k and b, and is thus independent of the randomization. It was indicated that a setting of $[k]_{\text{left}}$ leads to a conflict with R if and only if one of the inputs in C_k is linked to the output terminal of local label t in $[k]_{\text{left}}$. Therefore:

$$\Pr[x_k = 1] = \frac{\text{No. of settings of } [k]_{\text{left}} \text{ that link some input in } C_k \text{ to the output port } t \text{ of } [k]_{\text{left}}}{\text{total number of settings of } [k]_{\text{left}}},$$

$$\Pr[x_k = 1] = c_k \frac{\text{No. of settings of } [k]_{\text{left}} \text{ that link some fixed input in } C_k \text{ to the output port } t \text{ of } [k]_{\text{left}}}{\text{total number of settings of } [k]_{\text{left}}}.$$

In the case where the set S from which the left switch settings are selected is the entire set of the $q!$ one-to-one settings, it can be seen that

$$\Pr[x_k = 1] = \frac{c_k (q-1)!}{q!} = \frac{c_k}{q}.$$

If, on the other hand, the set S is limited as in the algorithm to $\{\pi_j \mid 0 \leqslant j \leqslant q-1\}$, then there is only one setting that links a fixed input x in C_k to the output port t because $\pi_j(x) = t$ is equivalent to $x + j \bmod q = t$ which, in turn, is equivalent to $j = [(t - x) \bmod q]$. Thus, under this limited case,

$$\Pr[x_k = 1] = c_k \frac{1}{q} = \frac{c_k}{q},$$

which is the same as above. This justifies why in the Switch Randomization Algorithm the setting choices were limited to $\{\pi_j \mid 0 \leqslant j \leqslant q-1\}$.

To summarize,

$$X = \sum_{k=0}^{p-1} x_k$$

is the sum of independent Bernoulli variables $(x_k)_{0 \leqslant k \leqslant p-1}$ but their parameters c_k/q are not necessarily identical. Therefore, X does not necessarily follow a binomial distribution. However, the mean of X can be computed and its standard deviation can be bounded. This will enable us to use the well-known Chebychev's inequality to derive useful probabilistic bounds on the communication delays of messages.

Theorem 4.3. Let μ_X and σ_x be the mean and standard deviation of X, respectively. Then, $\mu_X = 1$ and $\sigma_X < 1$.

Proof.

$$\mu_X = E(X) = \sum_{k=0}^{p-1} E(x_k) = \sum_{k=0}^{p-1} \frac{c_k}{q} = \frac{c}{q} = 1.$$

$$\operatorname{var}(X) = \sum_{k=0}^{p-1} \operatorname{var}(x_k) = \sum_{k=0}^{p-1} \frac{c_k}{q}\left(1 - \frac{c_k}{q}\right) < \sum_{k=0}^{p-1} \frac{c_k}{q} = \frac{c}{q} = 1.$$

Therefore, $\sigma_X = \sqrt{\operatorname{var}(X)} < 1$.

One immediate conclusion that can be drawn from the previous theorem is that the average value 1 of the delay X is $\leqslant$ the average of X under the Single Randomization Algorithm [where $1 \leqslant E(X) \leqslant 2$]. This gives an indication that this algorithm is better than the Single Randomization Algorithm on average.

More importantly, using Chebychev's inequality

$$\Pr[\mu_X - k\sigma_X < X < \mu_X + k\sigma_X] \geqslant 1 - \frac{1}{k^2},$$

it is concluded that:

$$\Pr[X \leqslant k] = \Pr[X < 1 + k] \geqslant \Pr[1 - k < X < 1 + k] \geqslant \Pr[\mu_X - k\sigma_X < X < \mu_X + k\sigma_X]$$

and therefore, $\Pr[X \leqslant k] \geqslant 1 - 1/k^2$. In particular, $\Pr[X \leqslant 5] \geqslant 0.96$.

The average value 1 of the delay X and the high probability that the delay is bounded by 5 reveals the power of the Switch Randomization Algorithm.

It should be noted that because of the very conservative Chebychev bound, the 0.96 figure does not fully exhibit the advantage of this algorithm over the Single Randomization Algorithm. However, the simulation analysis conducted in the next section will clearly show that advantage.

5. SIMULATION

We have simulated the three algorithms on Clos networks $C(p, q)$ for $4 \leqslant q \leqslant 32$, taking $p = q$ for convenience. For each algorithm and each value of q, several hundred arbitrary, randomly

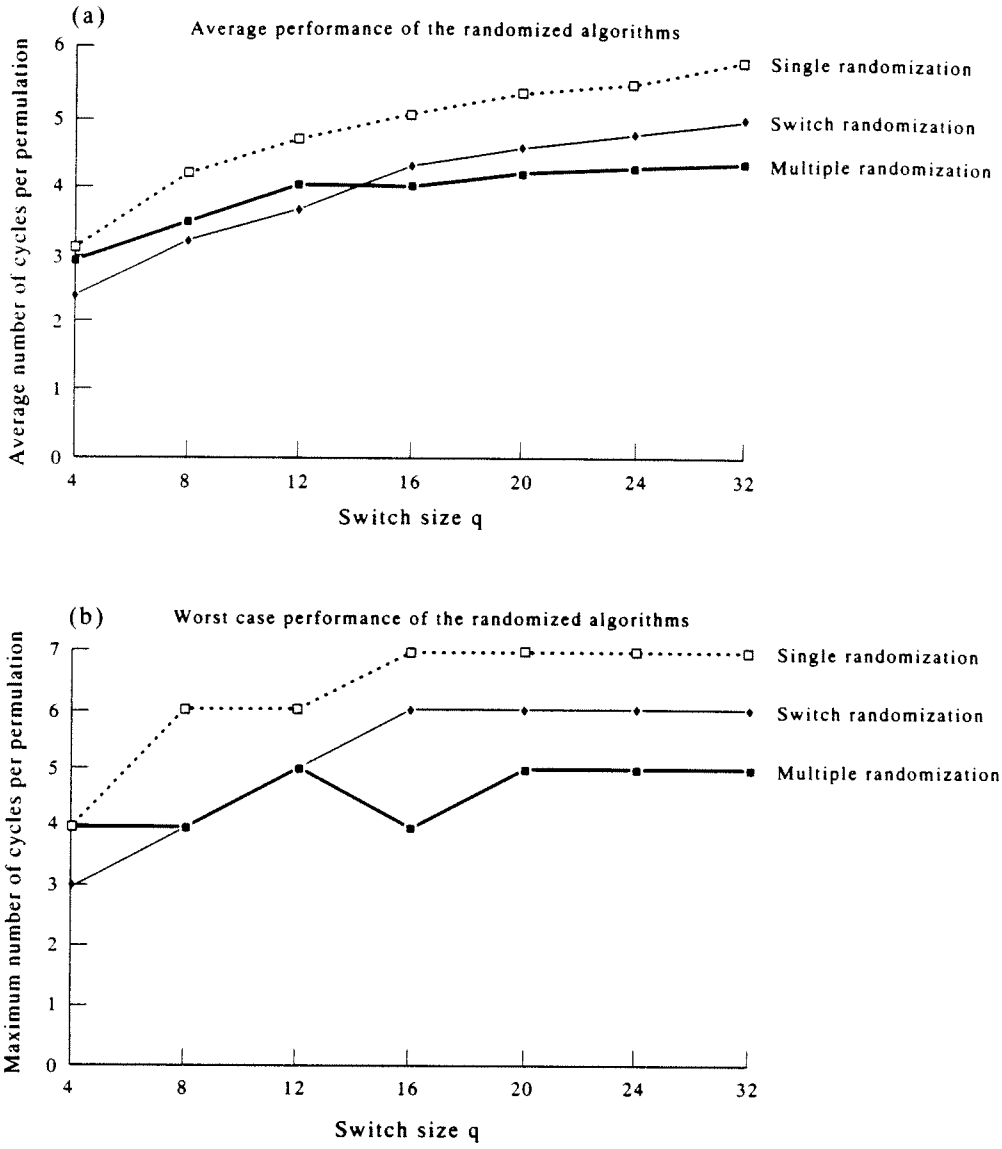

Fig. 2. Performance of the randomized algorithms.

selected permutations were routed. For each permutation we recorded the total number of cycles needed to complete routing the N messages of the permutation. For each algorithm and each value of q, we averaged the number of cycles per permutation over the several hundred routed permutations. We also measured the maximum number of cycles required by any of these routed permutations. Figure 2a plots the simulation findings of the average number of cycles per permutation in each of the three algorithms against the switch size q. Figure 2b plots the maximum number of cycles per permutation that we experienced in our simulations of the three algorithms for the various values of switch size q.

The plots clearly show that the three algorithms yield great performance, confirming the theoretical analysis and even giving better results. The average delay in any of the algorithms is 2.375–5.781 cycles/permutation, and the maximum experienced delay is 3–7 cycles/permutation. Comparing Fig. 2a and b shows that each algorithm exhibits a highly uniform delay as is evident from the fact that the difference between the average delay and maximum delay is about 1 cycle.

The algorithms are comparable in performance, differing by roughly 1–1.5 cycles/permutation. However, since permutations are routed frequently in parallel systems, a performance differential of 1 or 2 cycles (about 20 or 40% in our case) is significant. Therefore, the performance differential between the three algorithms is significant enough to warrant a closer comparison. The Switch Randomization Algorithm and the Multiple Randomization Algorithm are advantageous over the

Single Randomization Algorithm by more than one cycle. For switch sizes $q < 16$, the Switch Randomization Algorithm yields a somewhat smaller delay than the Multiple Randomization Algorithm. For larger switch sizes where $16 \leqslant q \leqslant 32$, which corresponds to machines of 256–1024 processors, the Multiple Randomization Algorithm is clearly the best of the three algorithms. It yields an almost deterministic average delay of 4–4.344 cycles/permutation and a maximum delay of 4–5 cycles/permutation. This very small, almost deterministic delay of permutations in medium-to-large Clos-connected systems makes this routing algorithm very advantageous and makes Clos networks very practical, efficient and desirable.

6. CONCLUSION AND FUTURE DIRECTIONS

This paper introduced and studied three randomized self-routing algorithms for Clos networks. The probabilistic analysis and the simulations conducted in this paper have shown that the communication delay of any permutation is very small. Both the Switch Randomization Algorithm and the Multiple Randomization Algorithm yield smaller delays than the straightforward Single Randomization Algorithm. For switch sizes <16 (i.e. for networks of size $N < 256$ processors), the Switch Randomization Algorithm performs better than the Multiple Randomization Algorithm by roughly half a cycle per permutation. In fact, under the Switch Randomization Algorithm, the delay is 2.375–4.281 cycles/permutation for switch sizes < 16. However, for switch sizes in the range 16–32 (i.e. N in the range 256–1024 processors), the Multiple Randomization Algorithm is the best of the three algorithms; it yields a delay of 4–4.344 cycles/permutation on average and a maximum delay of 4–5 cycles/permutation. It is a very small and almost deterministic delay that makes the algorithm highly useful.

These randomized routing algorithms enjoy several advantages. The first advantage is that the algorithms are very simple and easy to implement. The second advantage is that the algorithms take little computational time to set the switches due to the self-routed nature of randomized routing. Precisely, the only computation time needed is the small constant time of selecting a random number. The third advantage is that the communication delay of permutations when using these algorithms is extremely small. The fourth advantage is the universality of the algorithms in that they route all permutations.

The ease of implementation, the routing speed, the very low communication delay, and the universality make these algorithms superior to any other known universal routing algorithm for Clos or Benes networks. They also make Close networks superior to the nonuniversal banyan MINs. In addition, if these algorithms are used jointly with Youssef and Arden's approach in Ref. [9], every Omega-realizable permutation can be routed on Clos in a single cycle. Therefore, Clos networks can now deliver the same performance as banyan MIN δ networks on δ-realizable permutations, and deliver higher performance than δ networks on the permutations that cannot be realized on δ networks. These advantages and the comparative VLSI cost make Clos networks highly practical and attractive for large parallel systems.

Several research directions in randomized routing need to be pursued and are currently under investigation. First, the fault tolerance capabilities of the three algorithms are being examined. The algorithms turn out to be highly adaptable to fault tolerance and can tolerate many switch faults of various types. Second, other modes of routing than permuting will be studied probabilistically and using simulation. Finally, randomization will be applied to Clos–Benes networks of more than three columns to determine the efficacy of randomization in these networks as well as the effect of the number of columns on the communication delay.

REFERENCES

1. C. Clos, A study of non-blocking switching networks. *Bell Syst. Tech. J.* **32**, 406–424 (1953).
2. V. E. Benes, *Mathematical Theory on Connecting Networks and Telephone Traffic*. Academic Press, New York (1965).
3. T. Feng, A survey of interconnection networks. *Computer* **14**, 12–27 (1981).
4. D. K. Lawrie, Access and alignment of data in an arrary processor. *IEEE Trans. Comput.* **C-24**, 1145–1155 (1975).
5. M. C. Pease, The indirect binary n-cube multiprocessor array. *IEEE Trans. Comput.* **C-26**, 458–473 (1976).
6. C. Wu and T. Feng, On a class of multistage interconnection networks. *IEEE Trans. Comput.* **C-29**, 694–702 (1980).

7. D. C. Opferman and N. T. Tsao-Wu, On a class of rearrangeable switching networks, Part I: control algorithms. *Bell Syst. Tech. J.* **50,** 1579–1600 (1971).
8. G. F. Lev, N. Pippenger and L. G. Valiant, A fast parallel algorithm in permutation networks. *IEEE Trans. Comput.* **C-30,** 93–100 (1981).
9. A. Youssef and B. Arden, A new approach to fast control of $r^2 \times r^3$ 3-stage Benes networks of $r \times r$ crossbar switches. *Proc. 17th Int. Symp. Comput. Arch.*, Seattle, pp. 50–59 (1990).
10. J. Lenfant, Parallel permutations of data: a Benes network control algorithm for frequently used permutations. *IEEE Trans. Comput.* **C-27,** 637–647 (1978).
11. D. Nassimi and S. Sahni, A self-routing Benes network and parallel permutation algorithms. *IEEE Trans. Comput.* **C-30,** 332–340 (1981).
12. C. S. Raghavendra and R. Boppana, On self-routing in Benes and shuffle-exchange networks. *IEEE Trans. Comput.* **40,** 1057–1064 (1991).
13. K. Y. Lee, A new Benes network control algorithm. *IEEE Trans. Comput.* **C-36,** 768–772 (1987).
14. T. Feng and W. Young, An $O(\log^2 N)$ control algorithm. *Proc. Int. Conf. Parallel Processing*, pp. 334–340 (1985).
15. J. Beetem, M. Denneau and D. Weingarten, The GF11 supercomputer. *The 12th ann. Int. Symp. on Comput. Arch.*, pp. 108–113 (1985).

AUTHOR'S BIOGRAPHY

Abdou Youssef—Abdou Youssef received the B.S. degree in mathematics from The Lebanese University, Lebanon, in 1981, the M.A. and Ph.D. degrees in computer science from Princeton University, NJ, in 1985 and 1988, respectively. He has been an Assistant Professor in the Department of Electrical Engineering and Computer Science at The George Washington University, Washington, DC, since 1987. His research interests include interconnection networks and computer architecture, parallel processing, algorithms and fault tolerance. He has published numerous papers in journals and conference proceedings, and is involved in professional activities. Professor Youssef is a senior member of IEEE and a member of ACM.