$N \log N$ Dual Shuffle-Exchange Network with Error-Correcting Routing

Soung C. Liew, *Senior Member, IEEE* and Tony T. Lee, *Senior Member, IEEE,*

***Abstract*— This paper describes a dual shuffle-exchange switching network (DSN) that makes use of the principle of error-correcting routing. For motivation, parallels are drawn between error-correcting routing in switching and error-correcting coding in transmission. Based on a novel error-correcting and self-routing algorithm, we show by analysis and simulation that the DSN can achieve the Shannon's lower bound $N \log N$ on switch complexity while satisfying four desirable criteria: 1) self-routing property; 2) no queueing of packets at the inputs or inside the switch; 3) arbitrarily small packet-loss probability; 4) close-to-100% throughput. Different implementations of the basic DSN concept and their trade-offs are discussed.**

I. Introduction

At the fundamental level, a communication system consists of two complementary functions: transmission and switching. Transmission ensures that information is conveyed reliably from one location to another over a point-to-point channel, and switching allows information to be sent to different destinations at different times. To motivate the approach of our switch design, we wish to establish some parallels between the design philosophies of transmission and switching systems.

In the process of transmitting a message, noises are added to the message which are not intended by the sender. Shannon's work on Information Theory [1] establishes the interrelationship between the noise characteristics and the channel capacity. A major insight drawn from this theory is that instead of designing a channel to have extremely small bit-error rate, a more intelligent approach is to introduce error-correcting bits in addition to information bits in the message so that bit errors can be corrected at the receiver end. In this way, one avoids the technical difficulty of an error-free channel while still being able to achieve reliable transmission at a high rate.

In packet switching, the messages to be switched are divided into information blocks called packets. In a self-routing switch, the output address of a packet is embedded in the header of the packet. Typically, the switch consists of an interconnection of switch nodes [2]. Based on the header information, the switch nodes execute a distributed algorithm to route packets to their destinations without any external intervention. In such a system, packets may contend for the access of the same internal link or same output port. Whereas the design of a transmission system deals with ways to handle bit errors, the main design issue of a self-routing packet switch is how to handle packet contention. Routing errors are said to have occurred when packets are either dropped or delivered to the wrong destinations due to contention. Three switch-design philosophies for dealing with packet contention and routing errors are described below.

In Reference [3], we proposed a theoretical foundation for evaluation and comparison of a broad spectrum of packet switches within the framework of performance and complexity studies. We will follow the approach in this paper. Two of the more well-known switches proposed recently are the Batcher-banyan switch [4]–[6] and the Knockout switch [7]. The Batcher-banyan switch in [5] deals with packet contention by switching only a subset of input packets with non-conflicting output destinations in any given switch cycle. The packets that have been denied output access are buffered or queued at the inputs, so that they may try to reach their outputs again in the next switch cycle. For this reason, the Batcher-banyan switch has been classified as a *waiting system* in [3]. A packet is guaranteed to reach the correct destination after some waiting period. This strategy is analogous to operating a transmission channel to avoid errors totally, say, by using some ARQ (automatic repeat request) scheme [8]. The throughput of the input-queued Batcher-banyan switch is limited to below 60% [6],[9],[10], and the complexity is of order $N(\log N)^2$, which is above the Shannon's lower bound $N \log N$ on switch complexity [11].

Instead of solving the contention problem by packet waiting, the Knockout switch [9] allows up to L input packets to access any output simultaneously. If more than L packets arrive for the same output, excess packets are simply dropped, giving rise to the possibility of routing errors. However, the packet-loss probability can be made arbitrarily small by large enough L, which can be achieved by providing sufficient paths between inputs and outputs. Essentially, there is a trade-off between packet-loss probability and switch complexity [3]. This is the principle behind many switches that are classified as loss systems [3]. There is no mechanism for correcting routing errors since packets cannot be recovered once they are dropped. This strategy can be compared to designing a transmission channel to have very small bit-error rate. In transmission, as indicated by Information Theory, this may not be the most

Paper approved by M. S. Goodman, the Editor for Optical Switching of the IEEE Communications Society. Manuscript received September 9, 1991; revised May 30, 1992. This paper was presented in part at IEEE ICC '92, Chicago, June 1992.

The authors are with Department of Information Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong. This work was done while the authors were at Bellcore, New Jersey, U.S.A.

IEEE Log Number 9400962.

intelligent approach. In self-routing switching, no design that follows this approach is known to achieve the Shannon's lower bound $N \log N$ on complexity; to date, the lowest order of complexity is achieved by the dilated banyan network, and it is of order $N \log N(\log \log N)$ [3].

Within the class of loss systems, an alternative approach is to employ deflection routing. As a reference, both the tandem-banyan network [12] and the shuffle-exchange [see Section 2] network are based on deflection routing. In these systems, a packet that has lost contention in a switch node inside the overall switch is simply deflected to a wrong route temporarily. Redundancy is built into the routing process and the switch design so that the deflected packet can be routed in later switching stages in a way that corrects for the earlier mistake. This is analogous to using error-correcting codes to correct for bit errors in transmission channels.

Without the proper switch topology and routing algorithm that minimize the redundancy, the use of deflection routing in itself would not achieve the $N \log N$ lower bound switch complexity. For instance, the tandem-banyan network and the shuffle exchange network are both of order $N(\log N)^2$ [3]. This paper proposes a dual shuffle-exchange network with an adaptive distributed algorithm that achieves the $N \log N$ complexity. This network satisfies four desirable criteria for packet switches: 1) self-routing property; 2) no queueing of packets at the inputs or inside the switch; 3) arbitrarily small packet-loss probability; 4) close-to-100% throughput.

In the dual shuffle-exchange network, whenever a packet is deflected, two extra routing bits are attached to its existing routing tag in the header. The two routing bits indicate how the packet is to be routed later to correct for the deflection error. Thus, just as in transmission where we have error-correcting bits to correct transmission errors, we have error-correcting bits here to correct routing errors. The parallel between the basic idea of error-correcting routing and error-correcting coding is then clear. There is a significant but subtle difference, however: in switching, we know where and when a deflection occurs, whereas in transmission, the error bits are not known, otherwise correcting the error bits would be trivial. To better make use of this fact in switching, error-correcting bits for routing are introduced dynamically only when deflection occurs.

For background information, it is worth noting that the dual shuffle-exchange network is closely related to the crossback network in [13]. The crossback network is a shuffle-exchange network with bidirectional links. The evacuation time is defined to be the first time at which the expected link utilization in the system is less than $1/2N$, assuming no new packets are introduced after an initial set of packets are injected into the nodes. It was mentioned in [13] that the evacuation time of the crossback network can be shown to be of order $\log N$, although the actual proof was not given. This provides a clue that a unidirectional switching network with $N \log N$ complexity for any given packet-loss probability is achievable, and a simple "transformation" of the crossback network indeed leads to the construction of the dual shuffle-exchange network. Reference [13], however, concludes that the crossback network is inferior to other networks in practice although its evacuation time is asymptotically better, because of its high node complexity. On the other hand, the dual shuffle-exchange network is not only theoretically optimal, but also practically superior to many other switch designs, since the following important implementation issues are considered and examined in our construction:

1. An explicit self-routing algorithm and its variants are provided. These routing algorithms show that error-correcting routing can be implemented in a straightforward and distributed manner.
2. The high complexity of switch nodes in the crossback network, 25 crosspoints per node, can be reduced substantially. Specifically, we demonstrate that it is unnecessary to provide a nonblocking crossbar switch at each node, and that a banyan network with 4 crosspoints is sufficient.
3. Several design alternatives for the overall dual shuffle-exchange network are presented here. We show how the dual shuffle-exchange network can be configured as a $2N \times 2N$ network in practice to minimize complexity.
4. We observe that packets may experience routing deadlocks in the dual shuffle-exchange network. The same situation may also occur in the crossback network. We suggest several simple routing strategies to make the network deadlock-free.

To explain the details of the dual shuffle-exchange network, the remainder of this paper is organized as follows. Section II describes the shuffle-exchange network with deflection routing. Its deficiencies are pointed out and are used to motivate the design of the dual-shuffle exchange network. Section III provides the fundamentals of error-correcting routing in the dual shuffle-exchange network. Various implementation issues are addressed. The $N \log N$ bound is shown analytically to be achievable with the dual shuffle-exchange network in Section IV. The analysis and the correctness of the algorithm are verified by simulation. Section V deals with various miscellaneous system issues that deserve further attention and provides preliminary research results in these directions. Finally, the main results and implications of this work are summarized in Section VI.

II. Shuffle-Exchange Network with Deflection Routing

We consider a *shuffle-exchange network* (SN) with $N = 2^n$ inputs and outputs, and $L \geq n$ stages, each consisting of $\frac{N}{2}$ 2×2 switch elements. Figure 1 shows an 8×8 SN with 5 stages. To explain the self-routing mechanism of the SN, let us for the time being assume $L = n$; i.e., imagine Fig. 1 to have only 3 stages. We propose a labeling scheme to facilitate explanation. The switch nodes in each stage are labeled by an $(n-1)$-bit binary number from top to bottom. The upper input (output) of each node is labeled by 0, and the lower input (output) is la-

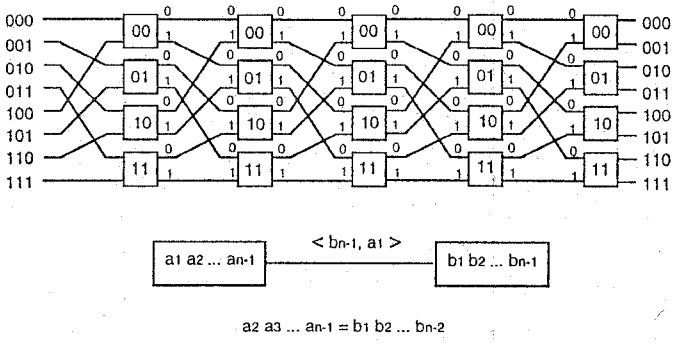

Fig. 1. An 8 × 8 shuffle-exchange network.

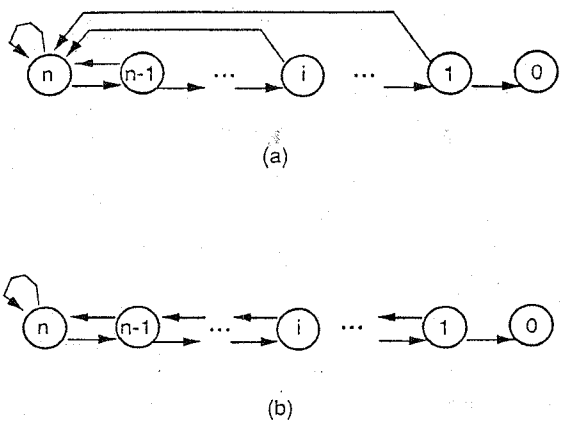

Fig. 2. (a) The state-transition diagram of a packet in the shuffle-exchange network, where the distance from destination is the state; (b) One-step penalty state transition.

beled by 1. A packet will be forwarded to output 0 (1) at stage i if the i^th most significant bit of its destination address is 0 (1). Two consecutive stages are connected by shuffle interconnection; node $A = (a_1 a_2 \cdots a_{n-1})$ is connected to node $B = (b_1 b_2 \cdots b_{n-1})$ of the subsequent stage iff $a_2 a_3 \cdots a_{n-1} = b_1 b_2 \cdots b_{n-2}$. The link between node A and node B will be labeled $< b_{n-1}, a_1 >$, where b_{n-1} is the output label of node A and a_1 is the input label of node B. The path of a packet from input to output is completely determined by the source address $S = s_1 \cdots s_n$ and the destination address $D = d_1 \cdots d_n$. This path can be expressed symbolically by using the above numbering scheme as

$$
\begin{aligned}
S = s_1 \cdots s_n & \\
\xrightarrow{<\emptyset, s_1>} \quad & (s_2 \cdots s_n) \xrightarrow{<d_1, s_2>} (s_3 \cdots s_n d_1) \\
\xrightarrow{<d_2, s_3>} \quad & \cdots \xrightarrow{<d_{i-1}, s_i>} (s_{i+1} \cdots s_n d_1 \cdots d_{i-1}) \\
\xrightarrow{<d_i, s_{i+1}>} \quad & (s_{i+2} \cdots s_n d_1 \cdots d_i) \\
\xrightarrow{<d_{i+1}, s_{i+2}>} \quad & \cdots \xrightarrow{<d_{n-1}, s_n>} (d_1 \cdots d_{n-1}) \\
\xrightarrow{<d_n, \emptyset>} \quad & d_1 \cdots d_n = D.
\end{aligned}
$$

The above sequence of nodes is embedded in the binary string $s_2 \cdots s_n d_1 \cdots d_{n-1}$ and can be revealed by an $(n-1)$-bit window moving one bit per stage from left to right.

The state of a packet traveling in the SN can be represented by a two-tuple (R, X), where R is the routing tag in the header of the packet and X is the label of the node that the packet resides. At the first stage, the packet is in state $(d_n \cdots d_1, s_2 \cdots s_n)$. The state transition is determined by the self-routing algorithm, and can be expressed by

$$(r_1 \cdots r_k, x_1 \cdots x_{n-1}) \xrightarrow{<r_k, x_1>} (r_1 \cdots r_{k-1}, x_2 \cdots x_{n-1} r_k).$$

Notice that the routing bit used is removed from the routing tag after each stage. At the final stage, the packet will reach the state $(d_n, d_1 \cdots d_{n-1})$, which is independent of the source address. That is, after this stage, a packet will arrive at the desired output regardless where it is coming from.

As an aside to clarify things, the location of a packet as described above actually refers to the location of the packet's header. At any one time, the information bits of the packet can be spread over several nodes and links along its path. This is because in actual implementation, the state of the switch node is set immediately upon the arrival of the header before the information bits have arrived.

So far we have considered only what happens when the routes taken by packets do not overlap. Contention occurs in a switch node when two input packets request the same output. One of them will be successfully routed while the other one will be deflected to the wrong output. Thus, if $L = n$, only a fraction of the packets can be routed correctly in the end. However, with $L > n$, we can implement a error-correcting routing algorithm as follows. All output links of the same row of stages n and above will be treated as belonging to the same logical address. Whenever a packet is deflected, its routing tag will be reset to $d_n \cdots d_1$, and the routing starts anew from the deflection point. With this strategy, some packets will reach their destinations after fewer numbers of stages than others, and a by-pass mechanism is needed to collect and multiplex packets that reach different physical links of the same logical address. The design of the by-pass mechanism is a hardware issue, and will be elaborated later after the fundamentals of error-correcting algorithm have been addressed. We see that a packet will eventually reach its destination address with good probability provided that the number of stages L is sufficiently large. The packet is lost, or dropped, if it can not reach the destination after L stages.

The error-correcting SN is highly inefficient, especially when n is large. This is because each time an error is made, routing of the packet must be restarted from the beginning. This fact is illustrated by the state-transition diagram in Fig. 2, in which the state is the distance or the number of stages away from destination. A desired network would be one with the state-transition diagram shown in Fig. 2(b), in which the penalty is only one step backward. We will discuss a network with a one-step error-correcting

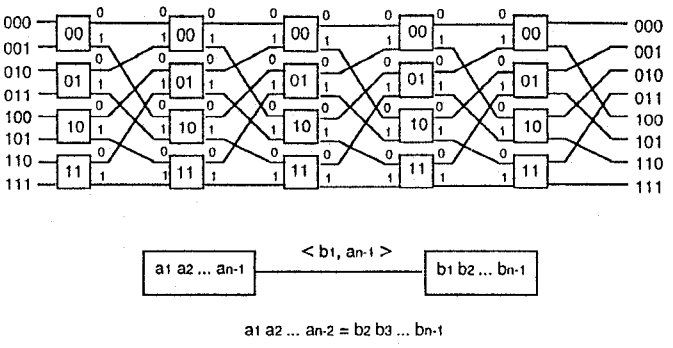

Fig. 3. An 8 × 8 unshuffle-exchange network.

routing algorithm in the next section.

III. Dual Shuffle-Exchange Network with Error-Correcting Routing

In this section, we introduce an error-correcting routing algorithm for a *dual shuffle-exchange network* (DSN) that reduces the penalty of deflection discussed in the previous section. The dual shuffle-exchange network consists of a shuffle exchange network and an *unshuffle-exchange network* (USN). An 8 × 8 USN is shown in Fig. 3. As can be seen, it is the mirror image of the SN. Routing bits used in successive stages proceed from the least significant bit to most significant bit rather than the other way round. Using a numbering scheme akin to that in SN, the path of a packet from input to output with source address $S = s_1 \cdots s_n$ and destination address $D = d_1 \cdots d_n$ can be expressed by

$$
\begin{aligned}
S = {} & s_1 \cdots s_n \\
& \xrightarrow{<\emptyset, s_n>} (s_1 \cdots s_{n-1}) \xrightarrow{<d_n, s_{n-1}>} (d_n s_1 \cdots s_{n-2}) \\
& \xrightarrow{<d_{n-1}, s_{n-2}>} \cdots \xrightarrow{<d_{i+2}, s_{i+1}>} (d_{i+2} \cdots d_n s_1 \cdots s_i) \\
& \xrightarrow{<d_{i+1}, s_i>} (d_{i+1} \cdots d_n s_1 \cdots s_{i-1}) \\
& \xrightarrow{<d_i, s_{i-1}>} \cdots \xrightarrow{<d_2, s_1>} (d_2 \cdots d_n) \\
& \xrightarrow{<d_1, \emptyset>} d_1 \cdots d_n = D.
\end{aligned}
$$

An $(n-1)$-bit window moving on the binary string $d_2 \cdots d_n s_1 \cdots s_{n-1}$ one bit per stage from right to left will reveal the sequence of nodes along the path. The initial state of the packet is $(d_1 \cdots d_n, s_1 \cdots s_{n-1})$, and the state transition is given by

$$
(r_1 \cdots r_k, x_1 \cdots x_{n-1}) \xrightarrow{<r_k, x_{n-1}>} (r_1 \cdots r_{k-1}, r_k x_1 \cdots x_{n-2})
$$

At the last stage, the packet is in state $(d_1, d_2 \cdots d_n)$. Note that shuffling precedes switching in each stage of the SN, but switching precedes unshuffling in each stage of the USN. Therefore, at the last stage of the USN, the packet would be considered as having reached its destination only after unshuffling of the output links.

Suppose an USN is overlaid on top of a SN, and each node in the USN is superimposed upon its corresponding

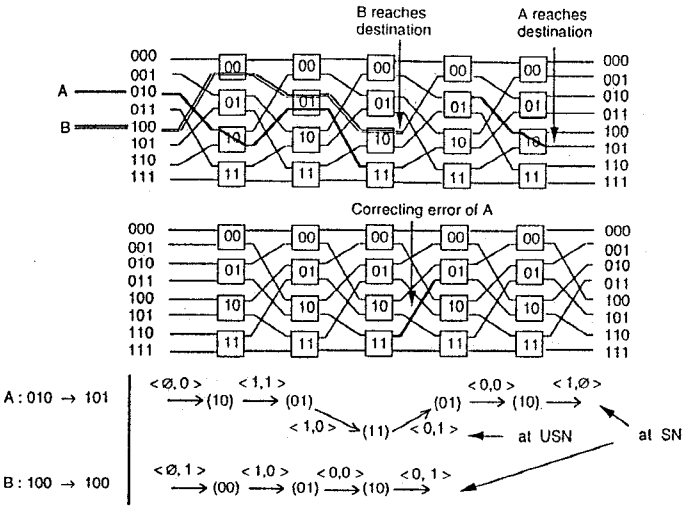

Fig. 4. An example of deflection error in SN being corrected in USN.

node in SN such that a packet in any one of these two companion nodes can access the outputs of both nodes. In such a network, the error caused by deflection in the SN can be corrected in the USN in only one step. This point can be best illustrated by an example. Suppose there are two packets A and B input to a SN as shown in Fig. 4, packet A, from input 010 to output 101, and packet B, from input 100 to output 100, will collide at the second stage when they arrive at node 01 and request for output 0. Suppose packet B wins the contention and packet A is deflected to node 11 in the third stage, Now, if we move packet A to the companion node 11 in the corresponding USN and switch it to output 0. Then it will reach node 01 at the next stage, the same node label when error occurred in the previous stage. At this point, the error caused by deflection has been corrected and packet A can return to its normal path in the SN again. Intuitively, this works because we can undo any routing operation in the SN by a reverse routing operation in the USN.

The above procedure can be formulated more rigorously as follows. Consider a packet in state $(r_1 \cdots r_k, x_1 \cdots x_{n-1})$, the packet should be sent out on link $< r_k, x_1 >$. Suppose it is deflected to link $< \bar{r}_k, x_1 >$ instead and reaches node $(x_2 \cdots x_{n-1} \bar{r}_k)$ in the next stage. If we attach the bit x_1 to the routing tag instead of removing the bit r_k, then the state of the packet will be $(r_1 \cdots r_k x_1, x_2 \cdots x_{n-1} \bar{r}_k)$ in the next stage. Now, we move the packet to the companion node in the USN to correct the error. If the packet is successfully routed this time, it will be sent out on link $< x_1, \bar{r}_k >$ and return to the previous state $(r_1 \cdots r_k, x_1 \cdots x_{n-1})$. Thus, the error has been corrected and the packet can return to the SN to complete its remaining journey. Similarly, the error occurred in the USN can also be fixed in one step in the SN. The state diagrams of this error-correcting procedure are illustrated in Fig. 5. The diagrams assume packets are deflected to links within their original networks. In general, a packet in the SN may

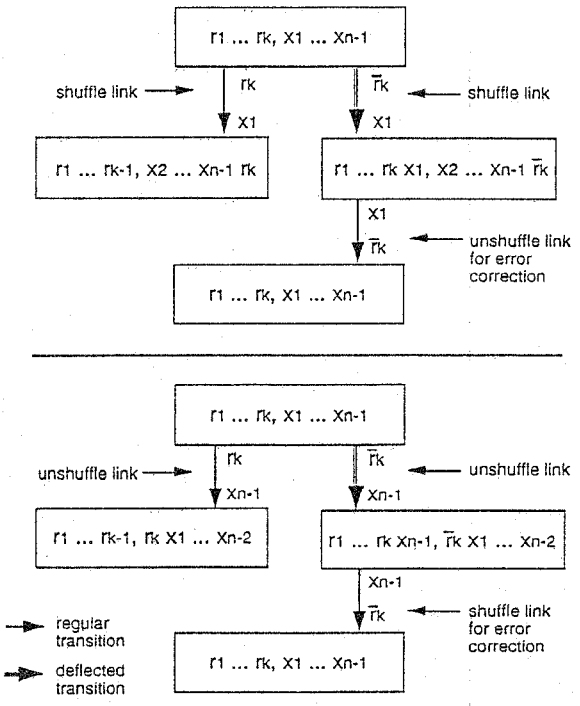

Fig. 5. State diagrams of the error-correcting procedure; top: routing in SN; bottom: routing in USN.

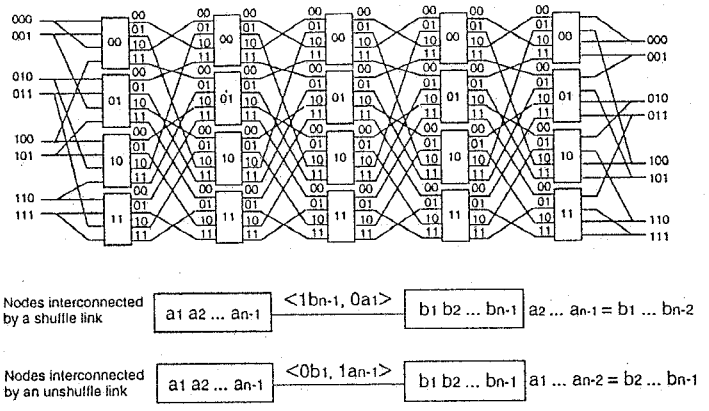

Fig. 6. An 8 × 8 dual-shuffle exchange network.

be deflected to the USN and vice versa, because of routing interference from other packets whose errors are being corrected. Furthermore, consecutive deflections can occur. The next subsection shows that both these considerations can be taken into account using a simple algorithm.

A. Error-Correcting Routing Algorithm

To allow transferring of packets from the SN to the USN, and vice versa, companion 2×2 switch elements in the SN and USN are merged to form 4×4 switch elements. A dual shuffle-exchange network built from 4×4 switch elements is shown in Fig. 6, where the SN and USN are combined to form the DSN using a new labeling scheme. The four inputs and outputs of a switch node are labeled by 00, 01, 10, 11 from top to bottom, where outputs 00 and 01 are connected to the next stage according to an unshuffling pattern, and outputs 10 and 11 are connected to the next stage according to a shuffling pattern. Specifically, a link with label $< 1a, 0b >$ is an unshuffle link and a link with label $< 0a, 1b >$ is a shuffle link. Two nodes $(a_1 \cdots a_{n-1})$ and $(b_1 \cdots b_{n-1})$ are connected by an unshuffle link $< 0b_1, 1a_{n-1} >$ if $a_1 \cdots a_{n-2} = b_2 \cdots b_{n-1}$, and by a shuffle link $< 1b_{n-1}, 0a_1 >$ if $a_2 \cdots a_{n-1} = b_1 \cdots b_{n-2}$. The construction of the link label is such that the second part of it can be used as the error-correcting routing bits in case of deflection. This will be elaborated shortly.

Since each switch node has four outputs, two routing bits are required to specify the desired output of a packet at each stage. A packet with destination $D = d_1 \cdots d_n$ can be either routed through the links of the USN or the links of the SN. Accordingly, the initial routing tag of a packet is set to either $0d_1 \cdots 0d_n$ or $1d_n \cdots 1d_1$, respectively.

The state of a packet at any particular time is $(c_1 r_1 \cdots c_k r_k, x_1 \cdots x_{n-1})$. There are two possible regular transitions at the present stage; the packet will be sent out on an unshuffle link if $c_k = 0$ and a shuffle link if $c_k = 1$. The corresponding state transitions are given by

$$(c_1 r_1 \cdots c_k r_k, x_1 \cdots x_{n-1}) \rightarrow$$

$$\begin{cases} \xrightarrow{<0r_k, 1x_{n-1}>} & (c_1 r_1 \cdots c_{k-1} r_{k-1}, r_k x_1 \cdots x_{n-2}) \\ & \qquad \text{if } c_k = 0 \\ \xrightarrow{<1r_k, 0x_1>} & (c_1 r_1 \cdots c_{k-1} r_{k-1}, x_2 \cdots x_{n-1} r_k) \\ & \qquad \text{if } c_k = 1 \end{cases}$$

Without deflections, it is easy to see that a packet with the initial routing set to $0d_1 \cdots 0d_n$ $(1d_n \cdots 1d_1)$ will stay in the USN (SN) links throughout the routing process until it reaches the desired destination at one of the USN (SN) links.

Let us consider how to deal with deflections. The error-correcting procedure can be demonstrated easily as follows. Suppose a packet in state $(c_1 r_1 \cdots c_{k-1} r_{k-1} 1 r_k, x_1 \cdots x_{n-1})$ is deflected, it may go out on any one of the other three outputs $0r_k, 0\bar{r}_k, 1\bar{r}_k$. We assume, without loss of generality, that the packet is routed to output link $< 0r_k, 1x_{n-1} >$, then it will arrive at the wrong node $(r_k x_1 \cdots x_{n-2})$ rather than the correct node $(x_2 \cdots x_{n-2} r_k)$ in the next stage. From the above state transition, the packet will return to its previous state if we attach the *error-correcting tag* $1x_{n-1}$ to the routing tag instead of removing $1r_k$. This point can be best illustrated by the following sequence of state transitions:

$$\begin{aligned} &(c_1 r_1 \cdots c_{k-1} r_{k-1} 1 r_k, x_1 \cdots x_{n-1}) \\ &\quad \xrightarrow{<0r_k, 1x_{n-1}>} (c_1 r_1 \cdots c_{k-1} r_{k-1} 1 r_k 1 x_{n-1}, r_k x_1 \cdots x_{n-2}) \\ &\quad \xrightarrow{<1x_{n-1}, 0r_k>} (c_1 r_1 \cdots c_{k-1} r_{k-1} 1 r_k, x_1 \cdots x_{n-1}) \end{aligned}$$

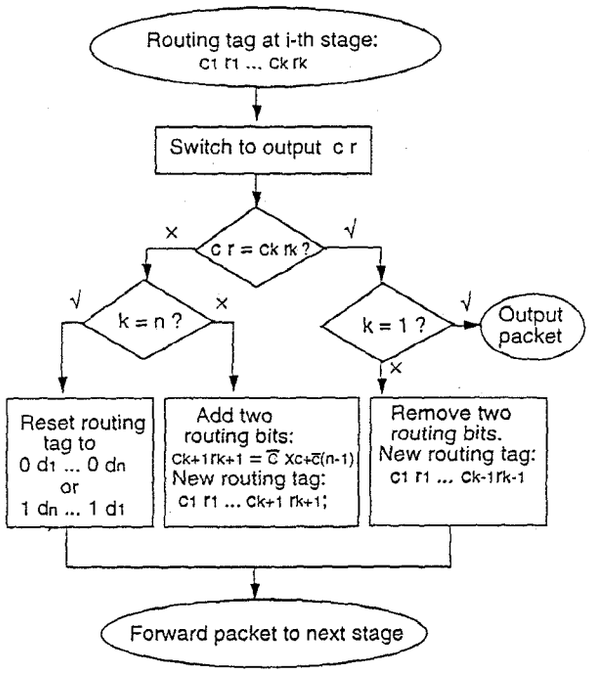

Fig. 7. The error-correcting routing algorithm.

Thus, the routing direction is simply given by the following:

1. If output $c_k r_k$ is available and $k > 1$, remove the two least significant bits from the routing tag and send the packet to the next stage.
2. If output $c_k r_k$ is available and $k = 1$, the packet has reached its destination; output the packet before the next shuffle if $c = 1$, and after the next unshuffle if $c = 0$.
3. If output $c_k r_k$ is unavailable and $k < n$, choose any one of the other available outputs, attach the error-correcting tag of the output to the routing tag and send the packet to the next stage.
4. If output $c_k r_k$ is unavailable and $k = n$, reset the routing tag to its original value, either $0d_1 \cdots 0d_n$ or $1d_n \cdots 1d_1$; this prevents the length of the routing tag from growing in an unbounded fashion.

Figure 7 illustrates the complete error-correcting algorithm. For any node with label $(x_1 \cdots x_{n-1})$, the error correcting tag of outputs 00 and 01 is $1x_{n-1}$, and the error-correcting tag of outputs 10 and 11 is $0x_1$. In other words, the second component $\bar{c}x$ in the link label $< cr, \bar{c}x >$ is the error-correcting tag of the output specified by the first component cr, and $x = x_{c+\bar{c}(n-1)}$, where $x_{c+\bar{c}(n-1)}$ is taken from the node label $(x_1 \cdots x_{n-1})$. Therefore, a packet deflected to link $< cr, \bar{c}x >$ will return to its previous state via link $< \bar{c}x, cr >$ in the next stage. This point is illustrated in Fig. 8 by the same example given in Fig. 4.

In the example, the deflection is immediately corrected in the next stage. In reality, deflections can occur in succession. It turns out that the algorithm in Fig. 7 is capable

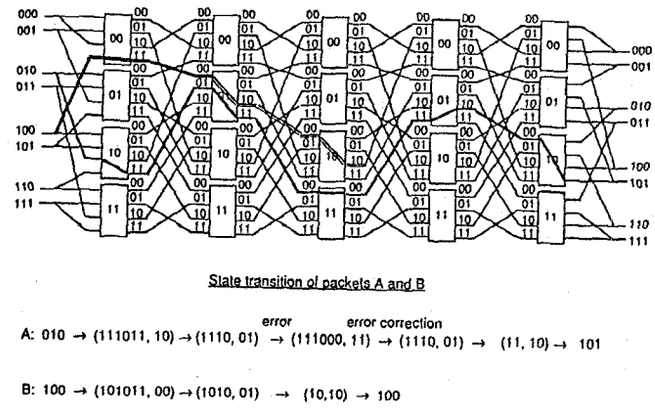

Fig. 8. An example of error-correcting routing in DSN.

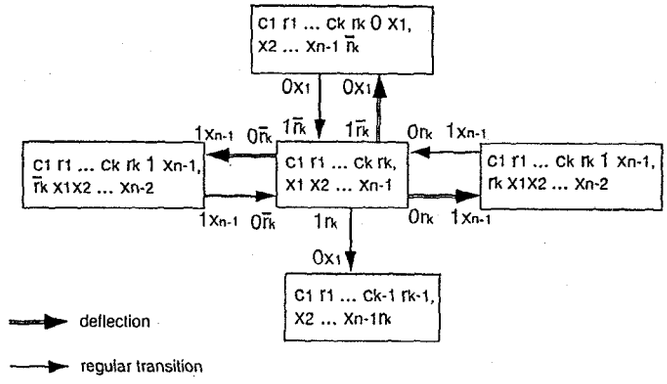

Fig. 9. Finite-state-machine representation of the error-correcting routing algorithm. assuming $c_k = 1$.

of dealing with consecutive errors implicitly. To see this, consider the finite-state machine representation of the algorithm as shown in Fig. 9. The state transitions when deflection occurred are given by

$$(c_1 r_1 \cdots c_k r_k, x_1 \cdots x_{n-1}) \rightarrow$$

$$\begin{cases} \xrightarrow{<0r, 1x_{n-1}>} & (c_1 r_1 \cdots c_k r_k 1 x_{n-1}, r x_1 \cdots x_{n-2}) \\ & \text{if } c_k r_k \neq 0r \\ \xrightarrow{<1r, 0x_1>} & (c_1 r_1 \cdots c_k r_k 0 x_1, x_2 \cdots x_{n-1} r) \\ & \text{if } c_k r_k \neq 1r \end{cases}$$

Suppose a packet is deflected from the current state, say state a, to one of the three possible states, say state b. The error-correcting algorithm guarantees that there is a transition through which the packet can return to state a later. But in case another deflection occurs immediately after the current deflection so that the packet then reaches state c, the algorithm simply attaches two additional error-correcting bits to the routing tag for correcting the new deflection. This provides a return transition from state c

to state b, from which the packet can eventually return to state a. We see that by attaching two error-correcting bits to the routing tag after each deflection, the way to correct the deflection is encoded into the routing tag until the deflection is finally corrected. Thus, the algorithm is recursive and capable of correcting consecutive errors.

B. Implementation Issues

The above basic concept can be employed in several network design alternatives. Figure 10(a) illustrates conceptually a design in which the DSN is configured to have dimensions $N \times N$. Both the SN and USN in the DSN are used for routing packets. An incoming packet is routed either to an SN input or an USN input, and accordingly, its routing tag is set to either $1d_n \cdots 1d_1$ or $0d_1 \cdots 0d_n$. In the first case, the packet's primary route is in the SN, and in the second case, the primary route is in the USN. Any excursion to the companion network is for error-correction purposes only. It should be noticed that in addition to ordinary routing, the primary network may also be called upon for error correction. This is because a packet can be deflected to either network regardless of its primary route, and deflections to the SN must be corrected in the USN, and vice versa.

Figure 10(b) shows an alternative design which configures the DSN as a $2N \times 2N$ switch. Here, the outputs $d_1 d_2 \cdots d_n$ of the SN and USN belong to two different logical addresses, $1d_1 d_2 \cdots d_n$ and $0d_1 d_2 \cdots d_n$, respectively. An incoming packet with destination $1d_1 d_2 \cdots d_n$ will be assigned the routing tag $1d_n 1d_{n-1} \cdots 1d_1$, and a packet with destination $0d_1 d_2 \cdots d_n$ will be assigned the routing tag $0d_1 0d_2 \cdots 0d_n$. This setup has the advantage that switch size is double that of Fig. 10(a).

For explanation purposes, however, we will concentrate on the design in Fig. 10(a) for the rest of this subsection. To further simplify the actual design, let us reexamine the SN and USN shown in Fig. 4. We note that the first shuffle of the inputs to the SN is unnecessary. The unshuffling pattern at the last stage of the USN can also be eliminated. However, in this case the initial routing tag for a packet in the USN must be set to $0d_n 0d_1 0d_2 \cdots 0d_{n-1}$. That is, a 2-bit cyclic right shift is performed on the original routing tag so that the two least significant bits $0d_n$ are used at the last stage of the new USN. The main advantage of the above modification is that the by-pass mechanisms within the SN and USN can both be implemented at the outputs of switch nodes; otherwise, the by-pass mechanism of the USN must be implemented at the inputs of the next stage (i.e., after unshuffling the output links of the present stage).

With the above modification, the block diagram of a switch node is shown in Fig. 11(a). Figure 11(b) depicts the connection of by-pass lines across two nodes of successive stages. The by-pass lines are needed only for stages n and above. The postprocessors process packets according to the logic outlined in Fig. 7. A packet that has reached its destination will be moved to the by-pass line and multiplexed with other packets that have reached the same destination in the previous stages. Note that although the mul-

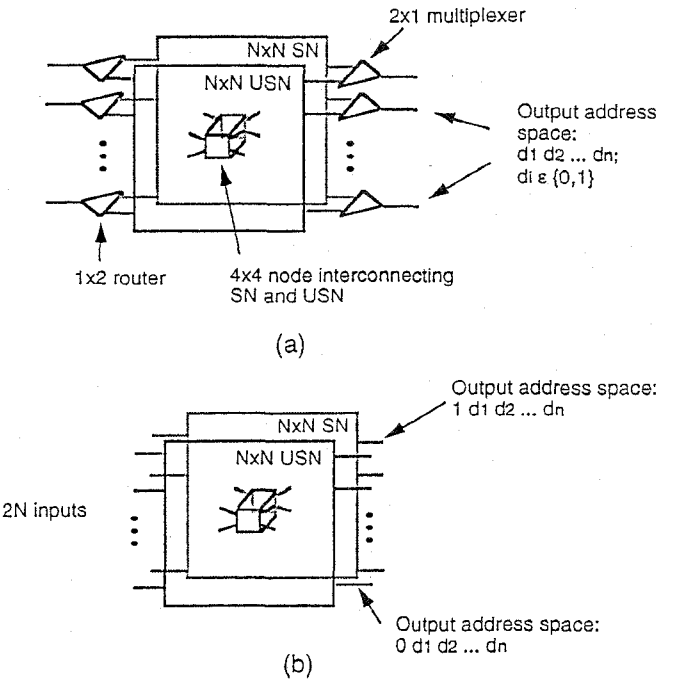

Fig. 10. (a) DSN configured as $N \times N$ switch; (b) DSN configured as $2N \times 2N$ switch.

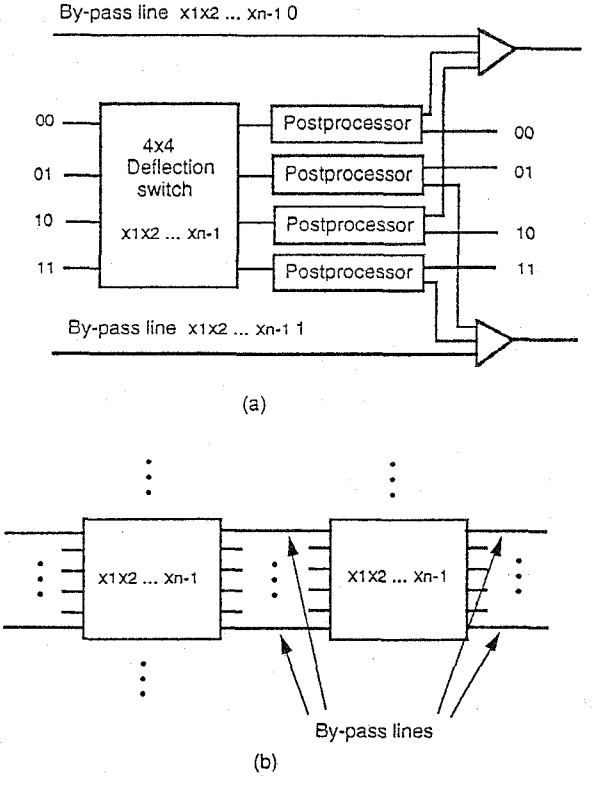

Fig. 11. (a) Block diagram of a switch node; (b) Interconnection of by-pass lines between adjacent switch nodes of same label.

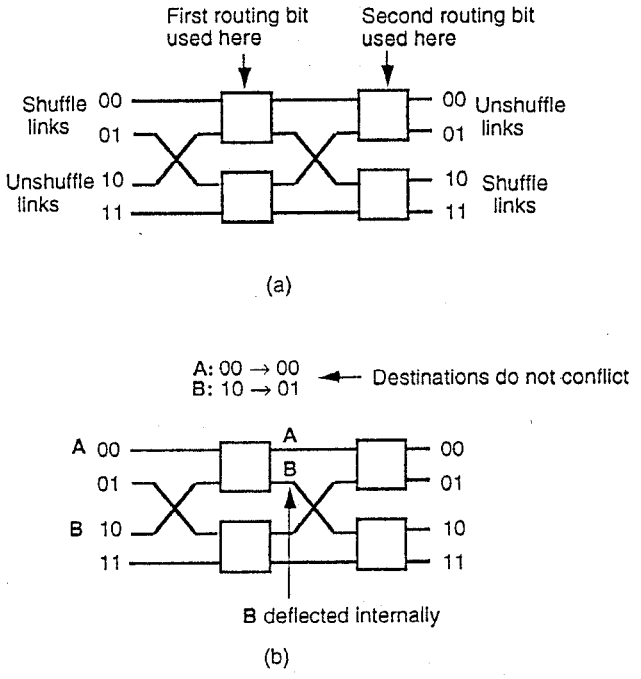

Fig. 12. (a) 4×4 banyan deflection switch; (b) An example of internal conflict when there is no output conflict.

tiplexers create physical queues internal to the network, the switch is logically an output-queued switch [9]; the physical buffers for the output queues are distributed across many switch nodes. Instead of the above, we may also have lines leading out of the switch from the by-pass locations and have a large-size multiplexer to multiplex packets for the same destination; in this case, there is only one physical queue for each output.

If we implement the 4×4 deflection switch using a nonblocking crossbar switch, a total of 16 crosspoints are needed. But using a nonblocking switch here would not exploit the principle of error-correcting routing to the fullest extent. Since the penalty of deflection is small, it is not necessary to implement "perfect switching" at the microscopic level. A 4×4 banyan network with 4 crosspoints as shown in Fig. 12(a) is good enough. There are two stages, and one of the two routing bits is used at each stage. A packet deflected in a 2×2 switch is marked. To reduce deflection probability at the second stage, priorities are given to unmarked packets if contention occurs. Granted that a packet may be deflected because of "internal conflict" even when there is no contending packet for the same "external output" (see Fig. 12(b)), the increase in the overall deflection probability is actually quite small. To see this, consider a simple estimate as follows: Suppose that all four inputs in the banyan network are loaded, and that each input packet is equally likely to be destined for any of the four outputs. Assume that under contention the winning (undeflected) packet will be chosen at random. The deflection probability with a nonblocking switch element is (input load − output load) / input load $= 1-(1-.75^4) \approx 0.3164$; the deflection probability with a banyan network is $1 - Pr$[not deflected at 1st stage] $\times$ Pr[not deflected at 2nd stage] $= 1 - 0.75(1 - 0.75 \times 0.25) \approx 0.3906$. We see that the additional deflection probability attributed to the blocking structure of the banyan network is only $0.3906 - 0.3164 = 0.0742$. Together with the fact that deflections can always be corrected later without severe penalties in the DSN, this means that it is probably better to use banyan switch elements rather than nonblocking switch elements to build the DSN. The nonblocking and banyan design alternatives will be compared in more detail in the next section.

IV. Analysis and Simulation

This section shows that the complexity of the DSN is of order $N \log N$ using an approximate analysis. Simulation results verifying the validity of the analysis and the correctness of the routing algorithm are then presented. Finally, the complexities of several implementation alternatives are compared.

A. Analysis: Complexity of DSN

Since routing requires at least $n = \log N$ stages, clearly the lower bound on the complexity of the DSN is $N \log N$. We will now show that $N \log N$ is also an upper bound on the order of complexity for a given packet-loss probability requirement P_{loss}. As a simplifying approximation, we assume that the input packets to a switch node are uncorrelated with each other and that they are equally likely to be destined for any of the four outputs. We further assume that the 4×4 switch elements are internally nonblocking. Let ρ_t be the load of an input at stage t, i.e., ρ_t is the probability that there is an incoming packet on an input link. Let p_t be the probability that a packet is successfully routed at stage t. We have

$$p_t = \frac{1 - \left(1 - \frac{1}{4}\rho_t\right)^4}{\rho_t}. \tag{1}$$

It is easy to show that $\frac{dp_t}{d\rho_t} \leq 0$ for $0 \leq \rho_t \leq 1$. Since the load cannot increase as t increases (number of packets cannot increase), ρ_t is a nonincreasing function of t. Therefore, $p_t \geq p_{t-1} \geq \cdots \geq p_1$. This agrees with our intuition that the probability of success increases as more and more packets are removed from the network. To obtain an upper bound on L, the number of stages required to meet a given P_{loss}, we perform a worst-case analysis in which p_t, for all t, is replaced by $p_1 = p$. That is, the L required by our system is bounded above by the L required in a corresponding "time-invariant" random walk depicted by the Markov chain in Fig. 13. The state of the Markov chain is the distance from destination, and $q = 1 - p$ is the deflection probability.

For analytical convenience, let us adopt a version of the DSN similar to the one in Fig. 10(a), but in which each of the N incoming packets is randomly routed to one of the $2N$ input ports of the SN and USN, with at most one packet assigned to the same input port. The input load on

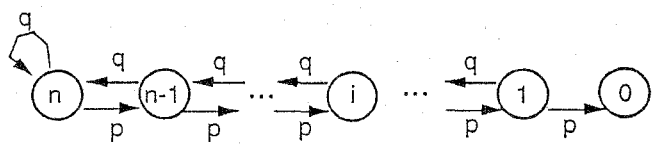

Fig. 13. Markov chain for bounding L analytically.

each input is $\rho_1 = 0.5$. This gives

$$p = p_1 = \frac{1-(1-0.5/4)^4}{0.5} \approx 0.828. \tag{2}$$

Let $g_i(k)$ be the conditional probability that a packet will reach its destination (or state 0) in k more steps given that its current state is i.

$$g_0(k) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

$$g_i(k) = pg_{i-1}(k-1) + qg_{i+1}(k-1);\ 0 < i < n, \tag{4}$$

$$g_n(k) = pg_{n-1}(k-1) + qg_n(k-1). \tag{5}$$

The z-transform $G_i(z) = \sum_{k=0}^{\infty} g_i(k)z^k$ is given by

$$G_0(z) = 1, \tag{6}$$

$$G_i(z) = pzG_{i-1}(z) + qzG_{i+1}(z);\ 0 < i < n, \tag{7}$$

$$G_n(z) = pzG_{n-1}(z) + qzG_n(z). \tag{8}$$

Equation (7) is a linear difference equation in terms of i; eqns. (6) and (8) are the boundary conditions. The general technique for solving the difference equation (7) is to substitute $G_i(z) = S^i(z)$. This gives

$$S^2(z) - \frac{1}{qz}S(z) + \frac{p}{q} = 0. \tag{9}$$

The roots of the quadratic equation are

$$S_1(z), S_2(z) = \frac{1}{2qz}\left(1 \pm \sqrt{1-4pqz^2}\right). \tag{10}$$

The general solution of $G_i(z)$ is

$$G_i(z) = C_1(z)S_1^i(z) + C_2(z)S_2^i(z). \tag{11}$$

The constants, $C_1(z)$ and $C_2(z)$, can be found by matching the boundary conditions at $i = 0$ and $i = n$ using eqns. (6) and (8). This yields

$$G_n(z) = \frac{pz[S_2^{n-1}(z)S_1^n(z) - S_1^{n-1}(z)S_2^n(z)]}{(1-qz)[S_1^n(z) - S_2^n(z)] - pz[S_1^{n-1}(z) - S_2^{n-1}(z)]}. \tag{12}$$

We can obtain a Chernoff bound on P_{loss} as follows:

$$\begin{aligned} P_{loss} &\le \sum_{k=L+1}^{\infty} g_n(k) \\ &\le \sum_{k=L+1}^{\infty} g_n(k)z^{k-(L+1)};\ \text{for some real } z \ge 1 \\ &\le z^{-(L+1)} \sum_{k=0}^{\infty} g_n(k)z^k \\ &= z^{-(L+1)}G_n(z). \end{aligned} \tag{13}$$

Thus, we are interested in $G_n(z)$ for real $z \ge 1$. It is clear from inequality (13) that to obtain a tight bound, $z^{-(L+1)}$ must be sufficiently small, or z sufficiently large. Let us determine a priori that we will choose z large enough that $S_1(\cdot)$ and $S_2(\cdot)$ are both complex. Then, it is more convenient to express them in the polar coordinates of the complex plane:

$$S_1(\theta), S_2(\theta) = \sqrt{p/q}\, e^{\pm j\theta}, \tag{14}$$

where $j = \sqrt{-1}$ and $\theta = \cos^{-1}\frac{1}{2z\sqrt{pq}}$. After some manipulation, we obtain

$$G_n(\theta) = \frac{(p/q)^{n/2}\sin\theta}{\sin(n+1)\theta - \sqrt{q/p}\sin n\theta}. \tag{15}$$

On examination, substituting $\theta = 0$ appears to give a reasonably tight Chernoff bound. Doing so yields

$$G_n(\theta = 0) = \frac{(p/q)^{n/2}}{(n+1) - n\sqrt{q/p}} \le \frac{(p/q)^{n/2}}{(n+1)(1-\sqrt{q/p})}. \tag{16}$$

Now, $\theta = 0$ implies $z = \frac{1}{2\sqrt{pq}}$. Substituting the above $G_n(\cdot)$ and z into inequality (13) and taking logarithms on both sides yields an upper bound for L:

$$L \le 2.793n - 3.554\ln(n+1) + 3.554\ln P_{loss}^{-1} + 1.162. \tag{17}$$

Since each stage consists of $N/2$ switch elements, the complexity of the DSN for a given P_{loss} is therefore of order $N \log N$.

B. Simulation Results

We have simulated several alternative DSN designs and collected their P_{loss} statistics. We have also experimented with several deterministic contention-resolution policies for the 4×4 nodes and found that routing deadlocks could occur quite easily for many of the policies. Details of deadlock issues are given in the next section. The simulation results presented here are based on a deadlock-free policy in which priorities are given to packets closest to their destinations.

Figure 14 plots P_{loss} versus L for various values of n. These are the results based on the DSN design in Fig. 10(a). That is, the DSN is used as an $N \times N$ switch, and an incoming packet is randomly routed to one of the two alternative input ports of the SN and USN. Note that this is slightly different from the assumption in the analysis, where the packet can be routed to any of the $2N$ input ports. This, however, should not give rise to drastically

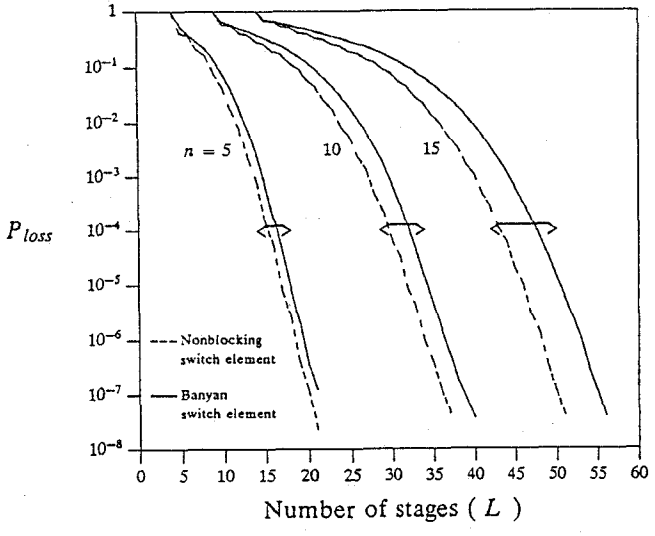

Fig. 14. P_{loss} versus L for various n; (dashed line: design using nonblocking 4 × 4 switch nodes; solid line: design using 4 × 4 banyan switch nodes).

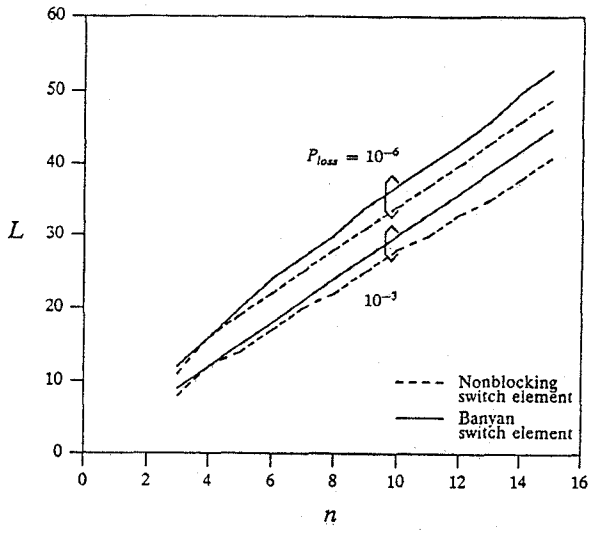

Fig. 15. L versus n, fixing P_{loss} at 10^{-3} and 10^{-6}; (dashed line: design using nonblocking 4 × 4 switch nodes; solid line: design using 4 × 4 banyan switch nodes).

different performance results. The dashed-line curves correspond to the design with 4 × 4 nonblocking switch nodes (see previous section). The solid-line curves assume 4 × 4 banyan switch elements instead (see Fig. 12). From the graph, we see that the L needed for a fixed P_{loss} in the second design is just slightly larger than that in the first design. Furthermore, for a given n, the difference in L is almost constant regardless of P_{loss} (for $P_{loss} \leq 10^{-2}$).

To examine the same result from a different angle, Fig. 15 plots L versus n for $P_{loss} = 10^{-3}$ and $P_{loss} = 10^{-6}$. As expected, and in agreement to our analysis, L is linear in n. We also see that the difference between the two designs is in the gradients of their lines. For the nonblocking-element design, the gradient is about 3.0, whereas for the banyan-element design, the gradient is about 3.2, a small difference indeed. Therefore, in terms of crosspoint count, the banyan-element architecture is less complex than the crossbar-element architecture by a factor of $\frac{3.2}{3.0} \times \frac{4}{16} = 0.27$. This observation and the fact that a more complex algorithm is needed to set the crossbar switch element clearly indicates that the banyan switch element is preferable.

Based on the banyan-element architecture, we next compare the designs of Fig. 10(a) and Fig. 10(b). To construct an $N \times N$ DSN using the second design, an $\frac{N}{2} \times \frac{N}{2}$ SN and an $\frac{N}{2} \times \frac{N}{2}$ USN are used. With this strategy and under full load, all input ports have one incoming packet. Therefore, it is expected that a larger L is needed to meet a given P_{loss}. Figure 16 plots L versus n for $P_{loss} = 10^{-6}$. The solid line corresponds to the design in Fig. 10(a) and the dashed line corresponds to the design in Fig, 10(b). Again, we see that the gradients of the linear curves are the difference between the two designs. From the complexity viewpoint, as long as the L required in the second design is less than twice that in the first design, then it is worth using the second design. This is because the "width" of the second design is half that of the first design. From the graph, we see that this is indeed the case.

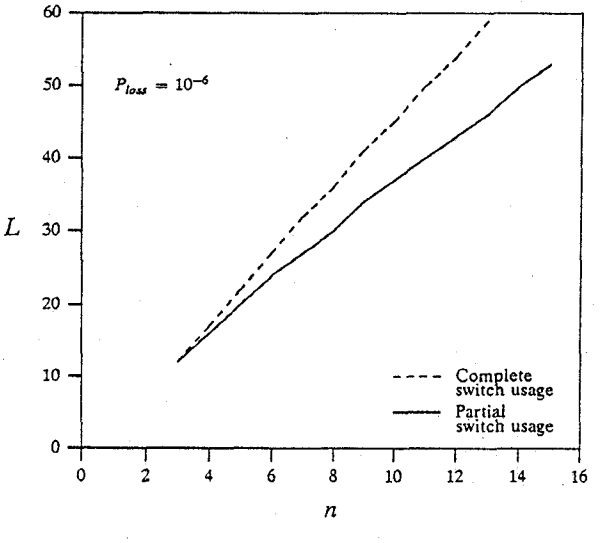

Fig. 16. L versus n, fixing P_{loss} at 10^{-6}, assuming 4 × 4 banyan switch nodes. (dashed line: using $N/2 \times N/2$ SN and USN to realize $N \times N$ DSN; solid line: using $N \times N$ SN and USN to realize $N \times N$ DSN).

V. Miscellaneous System Issues

There are several miscellaneous issues that deserve further attention. Some preliminary discussions are sketched in the following.

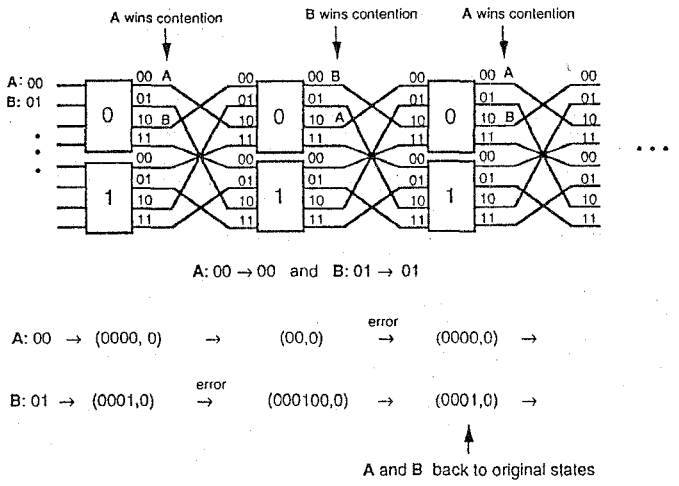

Fig. 17. A possible deadlock situation in a 4×4 DSN involving two packets.

A. Deadlock Prevention in DSN

Deadlocks could occur in routing when a group of packets contend with each other in a periodic fashion and they take turns winning the contention. This may give rise to situations in which their distances to destinations oscillate and fail to approach zero. As a result, no matter how large L is, P_{loss} can not be made arbitrarily small.

A deadlock example involving two packets is given in Fig. 17. In the 4×4 DSN, the source and destination addresses of packet A are both 00, the the source and destination addresses of packet B are both 01. As shown, both packets are destined for output 00 at the first stage. Suppose we adopt a policy of favoring packets on upper input ports when contention arises. Then, A wins the contention, and B is deflected to output 01. Two error-correction routing bits, 00, are then attached to the routing tag of B. The two packets arrive at the same node again at the next stage, both wanting to access output 00. But B wins contention this time because it is on the upper input port. Meanwhile, A will be deflected to output 01 and given two error-correction routing bits, 00. We see that the states of A and B after this stage are the same as their original states in the beginning. Thus, the contention pattern repeats itself indefinitely. More generally, deadlocks can involve more than two packets and can be quite subtle. Two simple ways to prevent deadlocks are now discussed.

Random Routing

An obvious way to avoid deadlock is to randomize the contention arbitration process in the switch nodes. When more than one packets want to access the same output, the winner will be drawn randomly. With probability one there will not be a deadlock, since the states of the packets cannot repeat themselves indefinitely. Our approximate analysis in the previous section assumes this policy.

Step-up Priority Routing

The simulations in the preceding section assume a priority scheme in which packets closest to their destinations are favored. Deadlocks will not occur with this policy either. To see this, suppose that a group of packets are involved in a deadlock. Let us concentrate on a packet that has the smallest distance to destination. Certainly, the distance of this packet will be decreased by one after each stage. Otherwise, it is losing contention to another packet with the same distance, and we will concentrate on this other packet instead. Eventually, we see that a packet will break free and reach distance 0 (i.e. its destination).

More generally, criteria other than the closest distance could be used. Let $p(A)$ denote the priority of packet A. The priorities may be described by a partial order $\geq$, whereby $p(A) \geq p(B)$ means packet A has a priority higher than or equal to that of packet B. In the same manner, $p(A) > p(B)$ means the priority of A is strictly higher than that of B. If each time a packet is successfully routed, its priority is raised by one one or more steps, then the policy is deadlock-free. The proof is essentially the same as before. Note that favoring packets with the longest distance does not fall under this scheme and therefore may not be deadlock-free.

A remaining research problem is whether there are other simple deadlock-free policies. It seems that the determination of whether a given policy is deadlock-free is a nontrivial problem in general.

B. A Class of Random-walk Models Realizable with DSN

The DSN operation assumed so far has an associated random-walk model as shown in Fig. 2(b). We now show that a rich class of random-walk models can actually be realized with the DSN.

State n in Fig. 2(b) is different from other states because error in this state does not increment the distance further; the routing tag is simply reset to the original routing tag. Suppose that this state is treated no differently than other states. The corresponding random-walk model is shown in Fig. 18(a), in which there is no reflecting barrier. The advantage of removing the boundary is that the number of by-pass locations in the DSN can be decreased. With this new random walk, regardless of the state of a packet, each deflection means the packet will need two more steps to reach its destination. In other words, a packet that experiences a total of k deflections will exit at stage $n+2k$ of the DSN. Therefore, only stages $n + 2k, k = 0, 1, 2...$, need to have by-pass mechanisms installed.

The disadvantage of the above scheme is that the length of the routing tag may in principle grow in an unbounded fashion. The random walk in Fig. 18(b) also only needs by-pass mechanisms at stages $n + 2k, k = 0, 1, 2...$, but bounds the routing tag to at most $2(n+1)$ bits. To realize this random walk, if an error is made in state n, the routing tag is reset to the original destination address plus two dummy bits: $0d_1 0d_2 \cdots 0d_n x_1 x_2$ or $1d_n 1d_{n-1} \cdots 1d_1 x_1 x_2$, where $x_1 x_2$ are the dummy routing bits. This moves the packet to state $n + 1$. In state $n + 1$, routing will be con-

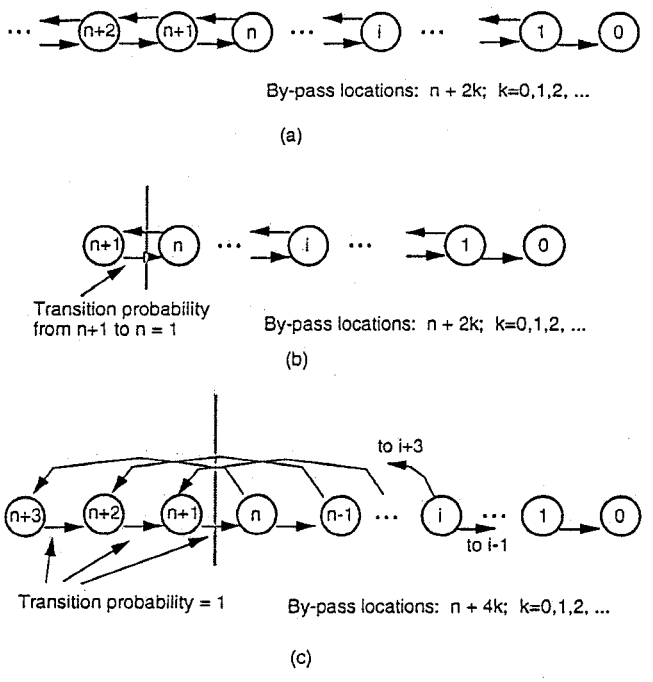

Fig. 18. Several random-walk models realizable with DSN: (a) Random walk without boundary; (b) Boundary with one-step reflection to the right; (c) 3-step penalty transition.

sidered successful regardless of the outcome and the two dummy routing bits removed. Consequently, a packet in state $n+1$ will move to state n in one step with probability 1. Thus, we have a boundary in the random walk that reflects the walk one step to the right.

In general, we could even further limit the by-pass locations to stages $n+2jk, k = 0, 1, 2, ...$ (for some fixed integer j). This is achieved by a random walk in which the penalty of deflection is $2j-1, j = 1, 2, ...$, steps backward. For illustration, an example with $j = 2$ is shown in Fig. 18(c). The by-pass locations for the corresponding DSN are at $n+4k, k = 0, 1, 2, ...$, since each left transition means four extra steps will be required to reach state 0. We first describe the modification on the routing algorithm when the packet is in state $i, i < n-2$. Imagine that deflections always happen in pair in the DSN, so that each time the packet is deflected, it will be deflected again the next time. Therefore, a total of four additional steps will be needed, two for the deflections and two for the corrections. We can "force" the above situation by introducing an additional one-bit field in the routing tag called *deflection balance*. Ordinarily, the value of this field is 0. When deflection occurs, two routing bits will be added according to the original routing algorithm, and the deflection-balance field will be set to 1. At the next stage, regardless of the routing result, the packet will be considered as having been deflected because its deflection-balance field is 1. Thus, two more routing bits will be added, again according to the original routing algorithm, and the deflection-balance bit set to 0. In this way, deflections are guaranteed to occur in pairs.

To complete the modification above, the deflections in states n, $n-1$, and $n-2$ must be treated differently because the state transitions cross the boundary at state n; when deflections occur in states $n-m$, $m = 0, 1$, or 2, the routing tag will be reset to the original routing tag plus $2(3-m)$ dummy routing bits. As in Fig. 18(b), routing in states above n will be considered successful regardless of the outcome.

It is easy to generalize the above scheme to $j = 3, 4, ...$ The deflection-balance field will simply be used to record the number of additional deflections that must be forced subsequent to the first deflection. However, as j increases, the penalty associated with deflection also increases. A larger L will then be needed in order to meet a fixed P_{loss}. Nevertheless, intuitively, as long a j is independent of n, the complexity of the DSN will still be of order $N \log N$, albeit the associated "constant" (i.e., the gradient of the L-versus-n curve) may be very large for large j. This statement remains to be substantiated. In addition, the practical trade-offs between reduced by-pass locations and increased L need to be studied more carefully.

C. Circuit Switching using DSN

Instead of using the DSN as a packet switch, one can also operate it as a circuit switch. In circuit switching, only one input is allowed to access an output at any given time. The multiplexer at a by-pass location is then replaced by a single line together with a mechanism to indicate whether the corresponding output is busy. To establish a new circuit, a probe signal containing the destination address is launched into the switch. It can be deflected by the existing circuits, but the same routing algorithms can be used to correct deflections. When the probe signal reaches its destination, the input will be informed whether the output is busy through a backward path running parallel to the forward path. If the output is idle, the circuit will then be accepted and it will use the path set by the probe signal.

In the context of circuit switching, the probability of blocking can be made arbitrarily small with sufficiently large L. A question of theoretical interest is how large should L be in order to totally eliminate circuit blocking given that no two circuits are destined for the same output. We conjecture that L is of order $\sqrt{N}$, although this remains to be confirmed by further research. In practice, however, L should be only of order $\log N$ to make the switch "seldom blocking".

IV. Conclusion

This paper has described a dual shuffle-exchange network that makes use of the principle of error-correcting routing. Based on a novel error-correcting algorithm, the dual shuffle-exchange network can achieve the Shannon's lower bound $N \log N$ on switch complexity [11] while satisfying four desirable criteria: 1) self-routing property; 2) no queueing of packets at the inputs or inside the switch; 3) arbitrarily small packet-loss probability; 4) close-to-100% throughput. As far as packet switches with the above prop-

erties are concerned, our present result is stronger than the $N \log N(\log \log N)$ complexity of the dilated-banyan network established previously by us. We have also conducted detailed implementation studies which demonstrate that the dual shuffle-exchange network is not only theoretically optimal, but also practically superior to many other switch designs.

On the whole, this work suggests that the principle of error-correcting routing is a powerful switch design technique. Intuitively, the dual shuffle-exchange network can achieve the lower bound on switch complexity because the penalty of routing errors is minimal. For further research, it will be interesting to investigate the possibility of other $N \log N$ error-correcting networks and the characteristics common to this class of networks. On the practical side, the combination of error-correcting routing with other switch design techniques, for example, dilation [3], will also be of much interest.

REFERENCES

[1] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, 1948, pp. 379-423 (Part I), pp. 623-656 (Part II).

[2] C. L. Wu and T. Y. Feng, *Tutorial: Interconnection Networks for Parallel and Distributed Processing* , IEEE Computer Society Press, 1984.

[3] T. T. Lee and S. C. Liew, "Broadband Packet Switches based on Dilated Interconnected Networks," to appear in *IEEE Transactions on Communications* .

[4] A. Huang and S. Knauer, "Starlite: A Wideband Digital Switch," *Conf. Record, IEEE Globecom 84* , pp. 121-125.

[5] Y. N. J. Hui, and E. Arthurs, "A Broadband Packet Switch for Integrated Transport," *IEEE Journal on Selected Areas in Communications* , Vol. 5, No. 8, October 1987, pp. 1264-1273.

[6] J. Giacopelli, M. Littlewood, and W. D. Sincoskie, "Sunshine: A High Performance Self-routing Broadband Packet Switch Architecture," *Proceeding of ISS'90* .

[7] Y. -S. Yeh, M. G. Hluchyj, and A. S. Acampora, "The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching," *IEEE Journal on Selected Areas in Communications* , Vol. 5, No. 8, October 1987, pp. 1274-1283.

[8] D. Bertsekas and R. Gallager, *Data Networks* , Prentice-Hall, 1987.

[9] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input vs. Output Queuing on a Space Division Packet Switch," *IEEE Transactions on Communications* , Vol. 35, No. 12, December 1987, pp. 1347-1356.

[10] S. C. Liew and K. W. Lu, "Comparison of Buffering Strategies for Asymmetric Packet Switch Modules," *IEEE Journal on Selected Areas in Communications* , Vol. 9, No. 3, April 1991, pp. 428-438.

[11] C. E. Shannon, "Memory Requirements in a Telephone Exchange," *Bell System Technical Journal* , Vol. 29, pp. 343-349, 1950.

[12] F. A. Tobagi, and T. Kwok, "The Tandem Banyan Switching Fabric: A Simple High-Performance Fast Packet Switch," *IEEE INFOCOM'91* .

[13] A. Krishna and B. Hajek, "Performance of Shuffle-Like Switching Networks with Deflection," *Proceedings of IEEE Infocom '90* , San Francisco, CA, pp. 473-480.

Soung C. Liew received the S.B., S.M., E.E., and Ph.D. degrees in electrical engineering from Massachusetts Institute of Technology, Cambridge, in 1984, 1986, 1986, 1988, respectively. From 1984 to 1988, he was a Research Assistant in the Local Communication Networks Group at the M.I.T. Laboratory for Information and Decision Systems, where he investigated fundamental design problems in high-capacity fiber-optic networks. He was also a Teaching Assistant for a graduate course on data communication networks.

In March 1988, he joined Bellcore, Morristown, New Jersey, where he has been a Member of Technical Staff in the Network Systems Research Laboratory. He is currently taking a leave of absence from Bellcore and is Senior Lecturer in the Chinese University of Hong Kong. He has conducted research and published actively in various areas related to broadband communications, including wavelength-division-multiplexed optical networks, high-speed packet-switch designs, system-performance analysis, routing algorithms, network-traffic control, and reliable and survivable networks.

His current research interests include interconnection networks, broadband network control and management, distributed and parallel computing, fault-tolerant networks, and optical networks. Dr. Liew is a senior member of the IEEE and a member of Sigma Xi and Tau Beta Pi.

Tony T. Lee received his B.S.E.E. degree from National Cheng Kung University, Taiwan in 1971, and his M.S. and Ph.D degrees in electrical engineering from Polytechnic University in New York, in 1976 and 1977, respectively. Currently, he is a Professor of Information Engineering at the Chinese University of Hong Kong.

He works in areas of broadband packet switch systems, performance analysis, parallel sorting networks, interconnection networks, relational database systems and protocol verification. Before joining Bellcore, he was with AT&T Bell Laboratories, Holmdel, NJ, from 1977 to 1983, He was an adjunct faculty member in the Electrical Engineering Department of Columbia University for the Fall term of the 1989 academic year. He also taught a pilot course on fast packet switches at MIT in 1990. From 1991 to 1993, he was a Professor of Electrical Engineering at Polytechnic University, Brooklyn, New York.

He is a member of Sigma Xi since 1977 and a senior member of IEEE since 1988. He was selected to be a Distinguished Member of Professional Staff by Bellcore since 1988. He is the recipient of the 1988 Leonard G. Abraham prize paper award from the IEEE Communications Society.