A New Self-Routing Multicast Network

Yuanyuan Yang, *Senior Member*, *IEEE*, and Jianchao Wang, *Member*, *IEEE*

Abstract—In this paper, we propose a design for a new self-routing multicast network which can realize arbitrary multicast assignments between its inputs and outputs without any blocking. The network design uses a recursive decomposition approach and is based on the binary radix sorting concept. All functional components of the network are reverse banyan networks. Specifically, the new multicast network is recursively constructed by cascading a binary splitting network and two half-size multicast networks. The binary splitting network, in turn, consists of two recursively constructed reverse banyan networks. The first reverse banyan network serves as a scatter network and the second reverse banyan network serves as a quasisorting network. The advantage of this approach is to provide a way to self-route multicast assignments through the network and a possibility to reuse part of network to reduce the network cost. The new multicast network we design is compared favorably with the previously proposed multicast networks. It uses $O(n \log^2 n)$ logic gates, and has $O(\log^2 n)$ depth and $O(\log^2 n)$ routing time where the unit of time is a gate delay. By reusing part of the network, the feedback implementation of the network can further reduce the network cost to $O(n \log n)$.

Index Terms—Multicast network, self-routing, binary radix sorting network, reverse banyan network, compact routing, recursive construction.

1 Introduction

Multicast or one-to-many communication is one of the most important collective communication operations [1] and is highly demanded in parallel and distributed applications as well as in other communication environments. For example, multicast is required to make updates in replicated and distributed databases and in commonly used parallel algorithms such as matrix multiplication and Fast Fourier Transform (FFT). Multiprocessor systems also require multicast for barrier synchronization and massage passing, and multicast is a critical operation for video/teleconference calls, video-on-demands services and distance learning in a telecommunication environment. Clearly, providing multicast support at hardware/interconnection network level is the most efficient way supporting such communication operations, [2], [3]. A switching network that can realize every multicast assignment between its inputs and outputs over edge-disjoint trees is referred to as a *multicast network*. This type of network has been investigated by several researchers in the literature [4], [5], [6], [7], [8], [9], [10].

In this paper, we design a new type of multicast network using an approach based on recursive decompositions of multicast networks. This approach was first introduced by Nassimi and Sahni [4] and was later adopted by Lee and Oruç [9] in their design of a multicast network. The $n \times n$ multicast network proposed in [4] uses $O(kn^{1+\frac{1}{k}} \log n)$ 2×2 switches and has $O(k \log n)$ depth and $O(k \log n)$ routing time for any k, $1 \leq k \leq \log n$.[1] Since the routing algorithm in [4] relies on a cube or a perfect shuffle connected parallel computer consisting of $O(kn^{1+\frac{1}{k}})$ processors, as mentioned in [9], the routing process actually takes $O(k \log^2 n)$ gate delays. Lee and Oruç [9] designed a multicast network with a special built-in routing circuit. Their network uses $O(n \log^2 n)$ logic gates, and has $O(\log^2 n)$ depth and $O(\log^3 n)$ routing time where the unit of time is a gate delay.

Another notable feature in our multicast network design is to adopt a self-routing scheme. Self-routing is a promising routing scheme which usually renders a network with faster switch setting and lower hardware complexity. However, most of self-routing network designs described in the literature are for permutation networks, for example, [11], [12], [13], [14]. In a recent work, Cheng and Chen [14] designed a new self-routing permutation network constructed by reverse banyan networks.

In this paper, we propose a design for a new self-routing multicast network, which is also based on the reverse banyan networks. We will explore the properties of a reverse banyan network so that it can be used to handle arbitrary multicast connections in a self-routing manner. Different from earlier proposed multicast networks, the new multicast network is conceptually simple and has good modularity. The network design is based on the binary radix sorting concept and all functional components of the network are recursively constructed reverse banyan networks. Therefore, it has a potential to greatly reduce the network cost by reusing part of the network. The new multicast network we design uses $O(n \log^2 n)$ logic gates, and has $O(\log^2 n)$ depth and $O(\log^2 n)$ routing time where the unit of time is a gate delay. Moreover, by reusing part of the network, the feedback version of our design can reduce the network cost to $O(n \log n)$.

- *Y. Yang is with the Department of Electrical and Computer Engineering, State University of New York, at Stony Brook, Stony Brook, NY 11794. E-mail: yang@ece.sunysb.edu.*
- *J. Wang is with GTE Laboratories, 40 Sylvan Road, Waltham, MA 02454. E-mail: jwang@gte.com.*

Manuscript received 3 Sept., 1997.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 105737.

1. k is a network parameter which equals $\frac{\log n}{\log m}$, and $(1, m)$-generator and $(n, \frac{n}{m})$-concentrator are components of the network.

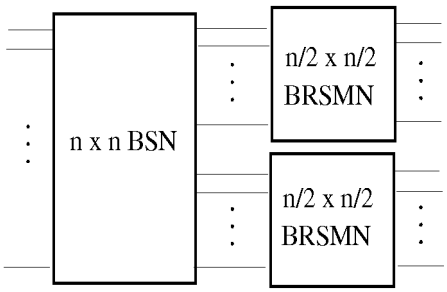

Fig. 1. The construction of an $n \times n$ binary radix sorting multicast network (BRSMN).

The rest of the paper is organized as follows: Section 2 introduces the binary radix sorting concept and gives the recursive definition of the new multicast network based on it. Section 3 describes a key component of the new multicast network, the binary splitting network. Section 4 discusses the basic building block of the binary splitting network, the reverse banyan network. Section 5 further explores the properties of the reverse banyan network as a scatter network and as a quasisorting network. Section 6 presents the distributed self-routing algorithms, and Section 7 discusses the implementation and complexity issues. Section 8 concludes the paper. Finally, Appendices A, B, and C give some detailed proofs and descriptions of the routing algorithms.

2 Multicast Networks Based on Binary Radix Sorting

We consider an $n \times n$ interconnection network with n inputs and n outputs where $n = 2^m$. Clearly, each input or output address can be expressed as an m-bit binary number $a_0 a_1 \ldots a_{m-1}$. For a multicast connection from network input i ($0 \leq i \leq n-1$) to a subset of network outputs, let I_i denote the subset of the outputs that input i is connected to. I_i is referred to as the *destination set* of the multicast connection, or simply the destination set of input i. Then, a multicast assignment can be expressed as a set $\{I_0, I_1, \ldots, I_{n-1}\}$ where, $I_i \cap I_j = \phi$ for $i \neq j$ and $\bigcup_{i=0}^{n-1} I_i \subseteq \{0, 1, \ldots, n-1\}$. For example, the following is a multicast assignment of an 8×8 network: $\{\{0,1\}, \phi, \{3,4,7\}, \{2\}, \phi, \phi, \phi, \{5,6\}\}$. We may also represent this multicast assignment in a binary format:

$$\left\{ \begin{Bmatrix} 000 \\ 001 \end{Bmatrix} \phi, \begin{Bmatrix} 011 \\ 100 \\ 111 \end{Bmatrix}, \{010\}, \phi, \phi, \phi, \begin{Bmatrix} 101 \\ 110 \end{Bmatrix} \right\}.$$

Clearly, a *permutation assignment* is a special case of a multicast assignment where each I_i has, at most, one element.

In this paper, we design a multicast network based on binary radix sorting concept and refer to it as a *binary radix sorting multicast network (BRSMN)*. An $n \times n$ BRSMN can be recursively constructed by an $n \times n$ *binary splitting network (BSN)* followed by two $\frac{n}{2} \times \frac{n}{2}$ BRSMNs which are directly linked to the upper half and the lower half of the outputs of the $n \times n$ BSN, respectively. The construction is shown in Fig. 1. For an input i, depending on its destination set we have four possible cases:

- Case 1: If all elements in its destination set I_i are in the upper half of the network outputs (i.e., the most significant bit of every binary address in I_i is 0), there will be a single connection from input i via the $n \times n$ BSN to an input of the upper $\frac{n}{2} \times \frac{n}{2}$ BRSMN with the same destination set I_i.
- Case 2: If all elements in I_i are in the lower half of the network outputs (i.e., the most significant bit of every binary address in I_i is 1), there will be a single connection from input i via the BSN to an input of the lower $\frac{n}{2} \times \frac{n}{2}$ BRSMN with the same destination set I_i.
- Case 3: If some elements in I_i go to the upper half and other elements in I_i go to the lower half (i.e., the most significant bits of the addresses in I_i contain both 0 and 1), there will be two connections from input i via the BSN. One goes to the upper $\frac{n}{2} \times \frac{n}{2}$ BRSMN, and another goes to the lower $\frac{n}{2} \times \frac{n}{2}$ BRSMN. The original destination set I_i will be split into two subsets which form the destination sets of the corresponding inputs of the upper and the lower $\frac{n}{2} \times \frac{n}{2}$ BRSMNs, respectively.
- Case 4: The destination set is empty (i.e., the input carries no message).

Then, for an $\frac{n}{2} \times \frac{n}{2}$ BRSMN, we also have four cases for each input, but this time we need to check the second most

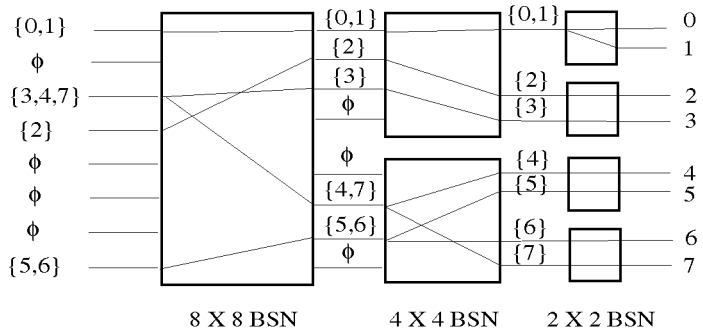

Fig. 2. A routing example for a multicast assignment in an 8×8 BRSMN.

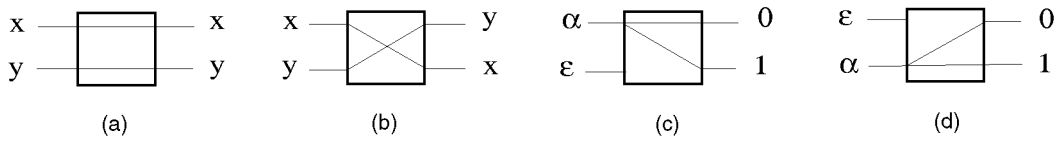

Fig. 3. Legal operations on the four values in a 2×2 switch, where $x, y \in \{0, 1, \alpha, \epsilon\}$. (a)Parallel. (b) Crossing. (c) Upper broadcast. (d) Lower broadcast.

significant bit of the binary addresses in the corresponding destination set, and so on. Finally, for a 2×2 BRSMN (i.e., a 2×2 switch), realizing a multicast or a unicast connection is straightforward. Fig. 2 shows the routing for the multicast assignment in the example mentioned earlier in an 8×8 binary radix sorting multicast network. From Fig. 2, we can also see that $n \times n$ BRSMN is constructed by an $n \times n$ BSN (level 1), followed by 2 $\frac{n}{2} \times \frac{n}{2}$ BSN's (level 2), then followed by 2^2 $\frac{n}{2^2} \times \frac{n}{2^2}$ BSN's (level 3), ..., and finally followed by $\frac{n}{2}$ 2×2 switches (level $\log n$). For more discussions on this type of multicast network structure, readers may also refer to [9].

Clearly, the problem of constructing a binary radix sorting multicast network (BRSMN) is now transformed to the problem of designing a binary splitting network (BSN) described above.

3 The Binary Splitting Network

As described in Section 2, the function of the binary splitting network is to split (when necessary) the multicast connection on each input based on whether each output in the destination set of the multicast connection belongs to the upper half or the lower half of the network outputs, and pass the (possibly split) multicast connections properly to the upper or lower $\frac{n}{2} \times \frac{n}{2}$ BRSMNs following it. To simplify the routing in a BSN, we use a routing tag with four values for each link: 0, 1, α, and ϵ, which correspond to Cases 1, 2, 3, and 4 described in the previous section, respectively, and are determined by the ith most significant bit of the multicast destination set on the inputs of BSNs at level i.

The operations on four values in a 2×2 switch are simply an extension to those on only two values 0 and 1. Fig. 3 shows all legal operations on four values in a 2×2 switch, where, the operations in Fig. 3a and Fig. 3b are unicast with no value changed, and the operations in Fig. 3c and Fig. 3d are broadcast with values α and ϵ on the inputs changed to 0 and 1 on the outputs.

In an $n \times n$ BSN using the four value routing tags, let n_0, n_1, n_α, and n_ϵ denote the numbers of inputs with value 0, 1, α, ϵ, respectively. Clearly, they satisfy some constraints. First, we have

$$n_0 + n_1 + n_\alpha + n_\epsilon = n. \tag{1}$$

Since at most a half of the outputs are in the upper half of the network outputs, we must have

$$n_0 + n_\alpha \leq \frac{n}{2} \text{ and } n_1 + n_\alpha \leq \frac{n}{2}. \tag{2}$$

Also, it is interesting to note that

$$n_\alpha \leq n_\epsilon, \tag{3}$$

which can be derived from (1) and (2).

Now, the function of a BSN is to transform the input tags 0s, 1s, αs, and ϵs to the output side such that all αs are eliminated, all 0s are in the upper half of outputs, all 1s are in the lower half of outputs. The numbers of the different tags on the outputs of the BSN should have the following relations: Let $\widehat{n_0}$, $\widehat{n_1}$, $\widehat{n_\epsilon}$, and $\widehat{n_\alpha}$ denote the numbers of outputs with value 0, 1, ϵ, and α, respectively. Since any α paired with one ϵ will be transformed to a pair of 0 and 1 by switch broadcast operations, we must have

$$\widehat{n_0} = n_0 + n_\alpha,\ \widehat{n_1} = n_1 + n_\alpha,\ \widehat{n_\epsilon} = n_\epsilon - n_\alpha,\ \text{and } \widehat{n_\alpha} = 0. \tag{4}$$

Having described the function of a BSN, we now consider the construction of the BSN. An $n \times n$ BSN can be constructed by cascading two $n \times n$ networks as shown in Fig. 4a. The first network is referred to as a *scatter network* which scatters all αs to 0s and 1s. In other words, the

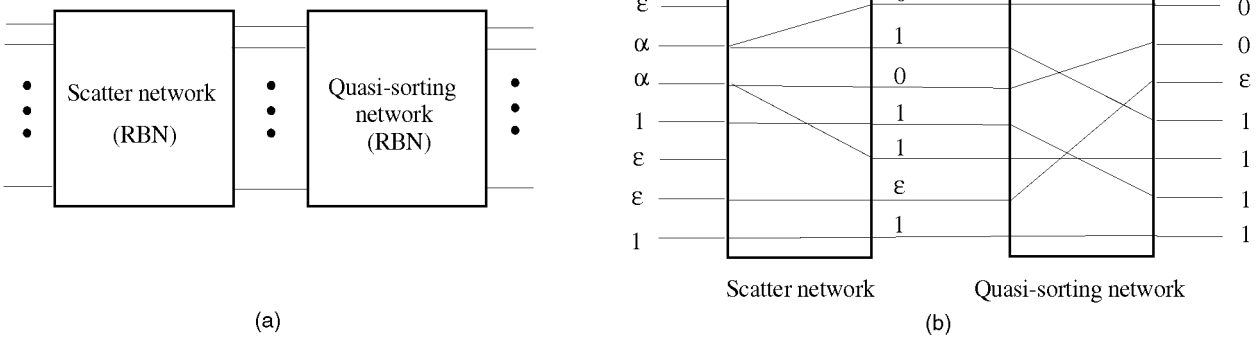

Fig. 4. Tags scattered in the first subnetwork and then quasisorted in the second subnetwork. (a) The construction of a binary splitting network (BSN). (b) An example of routing in a BSN.

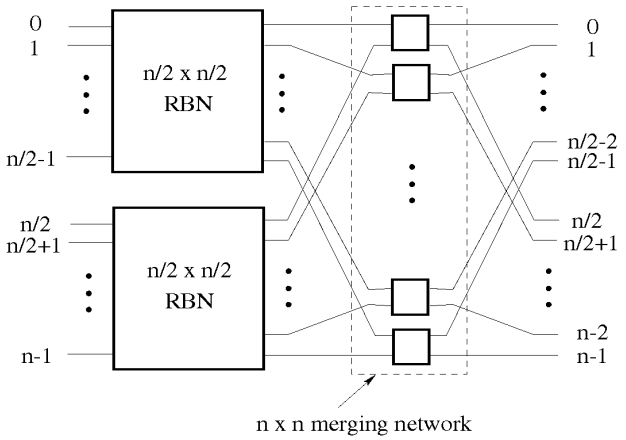

Fig. 5. The recursive definition of an $n \times n$ RBN. The dashed box is an $n \times n$ merging network.

transformation from the inputs to the outputs of the scatter network is

$$\{0, 1, \alpha, \epsilon\}^* \Rightarrow \{0, 1, \epsilon\}^*.$$

The second network is referred to as a *quasisorting network* which transfers all 0s to the upper half of the network outputs, and all 1s to the lower half of the network outputs, while each of ϵs may go to either the upper half or the lower half. It is easy to see that the network constructed by cascading a scatter network and quasisorting network can perform the functions of a BSN.

In Fig. 4b, we show how input tags in a BSN are scattered in the first subnetwork and then quasisorted in the second subnetwork.

In this paper, we use a reverse banyan network to implement both the scatter network and the quasisorting network in a BSN. As can be seen later, this implementation of a BSN not only leads to a faster self-routing multicast network but also provides a possibility to reuse part of the network to further reduce the network cost. In the following sections, we will mainly discuss how a reverse banyan network can perform the functions of a scatter network and a quasisorting network.

4 The Reverse Banyan Network

The reverse banyan network (RBN) used in our design is a variant of reverse banyan network, which was also used in Cheng and Chen's permutation network [14]. An $n \times n$ RBN is recursively constructed by two $\frac{n}{2} \times \frac{n}{2}$ RBNs followed by an $n \times n$ merging network. An $n \times n$ merging network consists of one stage of $\frac{n}{2}$ 2×2 switches, such that both the input and the output links of the stage are connected according to the perfect shuffle interconnection function [15]. The merging network merges the outputs of the upper and the lower $\frac{n}{2} \times \frac{n}{2}$ RBNs to form the outputs of the $n \times n$ RBN. The recursive construction of an $n \times n$ RBN, and an $n \times n$ merging network are shown in Fig. 5.

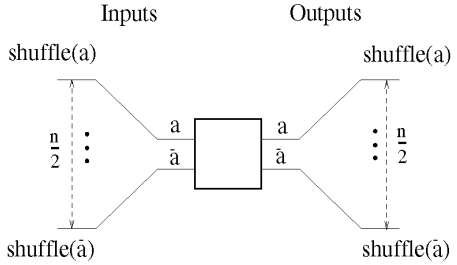

Fig. 6. The connections through a switch in a merging network, where $\bar{a} =$ exchange (a).

To facilitate the discussions on the functions of an RBN as a scatter network and as a quasisorting network, we need to introduce the following notations.

Suppose an n-bit sequence of two symbols, say, β and γ, is to be applied in parallel to the n inputs of a reverse banyan network. We define an n-bit *circular compact sequence* of two symbols βs and γs as follows:

$$C^n_{s,l;\beta,\gamma} = \begin{cases} \beta^{[s]}\gamma^{[l]}\beta^{[n-s-l]} & \text{if } s+l \le n \\ \gamma^{[l-n+s]}\beta^{[n-l]}\gamma^{[n-s]} & \text{if } s+l > n, \end{cases} \quad (5)$$

where $0 \le s < n$ and $0 \le l \le n$. The real meaning of $C^n_{s,l;\beta,\gamma}$ is that, in an n-bit sequence all l γ-bits are compacted together followed by also compacted $n-l$ β-bits in a circular way (modulo n), and s is the starting position for the γ-bit sequence. This concept is very important in this paper. In fact, the key results of this paper pertain to that under what conditions two circular compact sequences can be merged into a longer circular compact sequence. For example, the special circular compact sequence $C^n_{\frac{n}{2},\frac{n}{2};0,1} = 0^{[\frac{n}{2}]}1^{[\frac{n}{2}]}$ is what we expect after sorting tags 0s and 1s in a bit sorting network. In a later section, we will see that circular compact sequences will be used in merging and eliminating tags αs and ϵs in a scatter network.

Cheng and Chen [14] considered the circular compact sequence of 0s and 1s in their permutation network design, and found an interesting property which was used in their bit-sorting network. Since this property is also very useful for the designs of the scatter network and quasisorting network, we state it below in Theorem 1 in a more general form, and provide a different, much easier to understand proof for it. As can be seen later, the bit sorting network directly related to Theorem 1 will be used in the quasi-sorting network and the new technique used in the proof (i.e., Lemma 1) and some observations will be applied to the design of the scatter network.

Theorem 1. *For any β-γ values on the inputs of an RBN, a circular compact sequence with any starting position can be achieved at the outputs of the RBN under a proper setting for switches in the network.*

Suppose the inputs of the $n \times n$ RBN consist of l γs and $n-l$ βs, among which the upper half $\frac{n}{2}$ inputs contain l_0 γs and the lower half $\frac{n}{2}$ inputs contain l_1 γs, where $l_0 + l_1 = l$. Assume Theorem 1 holds for an $\frac{n}{2} \times \frac{n}{2}$ RBN. We will prove that it also holds for an $n \times n$ RBN by giving a positive answer to the following question.

Question 1. *Given integers n, s, l, l_0, and l_1 satisfying that n is an even number, $0 \le s < n$, $0 \le l \le n$, $0 \le l_0, l_1 \le \frac{n}{2}$, and $l = l_0 + l_1$, do there exist integers s_0 and s_1, $0 \le s_0, s_1 < \frac{n}{2}$,*

such that $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ and $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ can be merged to $C^{n}_{s,l;\beta,\gamma}$ through an $n \times n$ merging network (as defined in this section) under a proper switch setting (in a one-to-one mapping manner)?

Before we answer this question, we first review some observations on the shuffle/exchange interconnection functions. Consider an $n \times n$ merging network. Recall that the inputs (outputs) of the merging network are linked to switches according to the perfect shuffle function. Let a be a binary address of an input of a switch (with address $\lfloor \frac{a}{2} \rfloor$) in the network. Then *exchange*(a) denoted as $\bar{a}$ is the other input of the switch. The inputs of the merging network with addresses *shuffle*(a) and *shuffle*$(\bar{a})$ are linked to inputs a and $\bar{a}$ of the switch, respectively, and the outputs a and $\bar{a}$ of the switch are linked to the outputs of the merging network with addresses *shuffle*(a) and *shuffle*$(\bar{a})$, respectively. (See Fig. 6, and the definition of shuffle/exchange functions in [15].) We can see that connections are only possible between the inputs and the outputs of a merging network with addresses *shuffle*(a) and *shuffle*$(\bar{a})$. Since only one-to-one connections are considered here, the switch has only two settings: parallel and crossing. The parallel setting corresponds to the connections from input *shuffle*(a) to output *shuffle*(a) and from input *shuffle*$(\bar{a})$ to output *shuffle*$(\bar{a})$ (see Fig. 7a); while the crossing setting corresponds to the connections from input *shuffle*(a) to output *shuffle*$(\bar{a})$ and from input *shuffle*$(\bar{a})$ to output *shuffle*(a) (see Fig. 7b). Note that $|shuffle(a) - shuffle(\bar{a})| = \frac{n}{2}$, that is, two inputs $\frac{n}{2}$ apart in their addresses can be connected to outputs with the same addresses in a parallel or crossing way as shown in Fig. 7a and Fig. 7b, respectively.

Let r_i denote the setting for switch i. Let $r_i = 0$ if the switch is set to parallel, and $r_i = 1$ if the switch is set to crossing. Thus, the switch setting of the $\frac{n}{2}$ switches in the merging network is an $\frac{n}{2}$-bit sequence of 0s and 1s. Similar to the definition of circular compact sequence in (5), we use $W^{\frac{n}{2}}_{s,l;0,1}$ to specify the compact switch setting of $\frac{n}{2}$ switches in an $n \times n$ merging network, where l consecutive switches have setting 1 with starting position s, and the rest of switches have setting 0 in a circular way. In the case of a switch can also be set to upper broadcast and lower broadcast as shown in Fig. 3c and Fig. 3d, for any switch i,

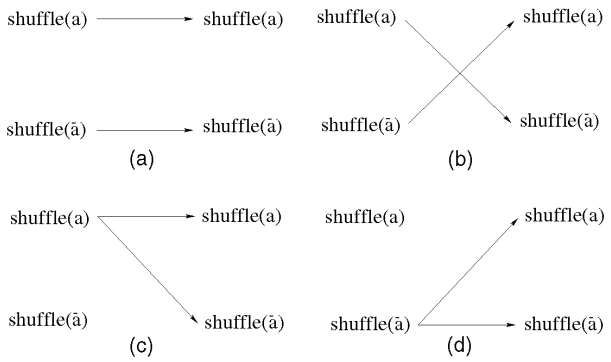

Fig. 7. Four different switch settings for a 2×2 switch in an $n \times n$ merging network. (a) Parallel. (b) Crossing. (c) Upper broadcast. (d) Lower broadcast.

in addition to switch settings $r_i = 0$ or 1, let the switch setting $r_i = 2$ if the switch is set to upper broadcast, and $r_i = 3$ if the switch is set to lower broadcast. We can extend the binary circular compact switch setting to trinary. We explain the extension by the following example. A trinary circular compact sequence of switch setting $W^{\frac{n}{2}}_{s,l_1,l_2;\beta_1,\beta_2,\beta_3}$ means l_1 consecutive β_2s followed by l_2 consecutive β_3s and then followed by $\frac{n}{2} - l_1 - l_2$ β_1s in a circular way, with s being the starting position of the β_2s sequence.

The following lemma gives a positive answer to Question 1.

Lemma 1. *Given integers n, s, l, l_0, and l_1 satisfying that n is an even number, $0 \le s < n$, $0 \le l \le n$, $0 \le l_0, l_1 \le \frac{n}{2}$, and $l = l_0 + l_1$. Let*

$$s_0 = s \bmod \frac{n}{2},\ s_1 = (s + l_0) \bmod \frac{n}{2}$$

and the switch setting of an $n \times n$ merging network be $W^{\frac{n}{2}}_{0,s_1;\bar{b},b}$, where

$$b = \left[(s + l_0) \ \mathbf{div}\ \frac{n}{2} \right] \bmod 2,$$

and $\bar{b} = (1 - b) \bmod 2$. Then, $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ and $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ can be merged to $C^{n}_{s,l;\beta,\gamma}$ through the $n \times n$ merging network under the above switch setting.

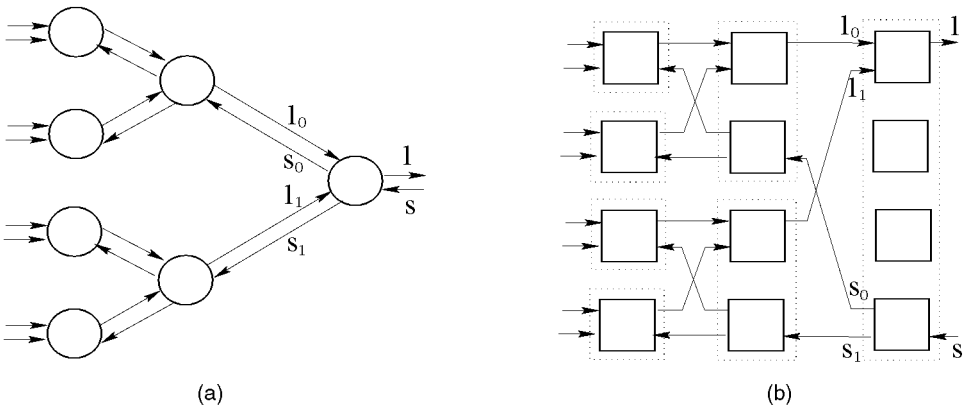

Fig. 8. The binary tree structure of an RBN. (a) The forward and backward phases in the binary tree. (b) The forward and backward trees embedded in a reverse banyan network.

Proof. See Appendix A. □

Lemma 1 is actually the inductive step of the proof of Theorem 1, and it is easy to check that for $n = 2$ Theorem 1 holds. Hence, Theorem 1 is proved.

Note that for a full permutation assignment, only two tag values 0 and 1 are used. Let β be 0, γ be 1, and the initial starting position for an $n \times n$ RBN be $s = \frac{n}{2}$. Clearly the total number of 1s in this case is $l = \frac{n}{2}$. By Theorem 1, the circular compact sequence $C^n_{s,l;0,1} = 0^{[\frac{n}{2}]}1^{[\frac{n}{2}]}$, which represents the function of bit sorting (in an ascending order), can be achieved by a proper switch setting.

5 Constructing the Binary Splitting Network by the RBNs

Recall that a binary splitting network consists of a scatter network and a quasisorting network. In this section, we show how to use the same basic component network, a reverse banyan network, to perform the function of a scatter network and the function of a quasisorting network, respectively.

5.1 The Reverse Banyan Network as a Scatter Network

In the last section, we considered only two tag values (0 and 1) for a full permutation assignment in an RBN. Now, for a multicast assignment in an RBN, we must deal with four tag values 0, 1, α, and ϵ. Recall that the function of a scatter network is to split the αs on its input into 0s and 1s on its output.

The following theorem gives one of the main results of this paper, which states that the function of a scatter network can be achieved under a proper switching setting. The proof of this theorem is given at the end of this subsection.

Theorem 2. *Given an $n \times n$ RBN, which is used as the scatter network of an $n \times n$ BSN, with 0, 1, α and ϵ values on the inputs. Let n_0, n_1, n_α, and n_ϵ be the numbers of inputs with value 0, 1, α, and ϵ, respectively. Then under a proper setting for switches in the network, αs can be eliminated at the outputs of the RBN, i.e. the outputs of the RBN have only values 0, 1, and ϵ with*

$$\widehat{n_0} = n_0 + n_\alpha,\ \widehat{n_1} = n_1 + n_\alpha,\ \widehat{n_\epsilon} = n_\epsilon - n_\alpha,\ \text{and}\ \widehat{n_\alpha} = 0,$$

where $\widehat{n_0}$, $\widehat{n_1}$, $\widehat{n_\epsilon}$, and $\widehat{n_\alpha}$ are the numbers of outputs with value 0, 1, ϵ, and α, respectively, and satisfying that $0 \le \widehat{n_0}, \widehat{n_1} \le \frac{n}{2}$ and $\widehat{n_0} + \widehat{n_1} + \widehat{n_\epsilon} = n$.

It is worth pointing out that although $n_\alpha \le n_\epsilon$ holds globally for the original $n \times n$ RBN which is the scatter network of an $n \times n$ BSN (see (3)), for the recursively defined subnetwork $n' \times n'$ RBN of the $n \times n$ RBN, we may have $n'_\alpha \ge n'_\epsilon$, where n'_α and n'_ϵ are the numbers of inputs (of this $n' \times n'$ RBN) with values α and ϵ, respectively. This is because that αs and ϵs are distributed nonuniformly.

To simplify the problem, we can combine 0 and 1 into a single value χ. A link has a value χ if it has a single value 0 or 1. If the two inputs of a 2×2 switch have values α and ϵ respectively, α can be scattered so that the two outputs of the switch have values χs. By exploring the properties of the circular compact sequence, we can obtain the following general results for an $n \times n$ RBN with any numbers of αs and ϵs on its inputs. This result will be used in the proof of Theorem 2.

Theorem 3. *For any values on the inputs of an $n \times n$ RBN, let n_α, and n_ϵ be the numbers of inputs with value α and ϵ, respectively. Clearly, $n_\alpha + n_\epsilon \le n$, and the rest of the inputs of the RBN have values χs. Let s be any integer such that $0 \le s < n$. Then,*

1. *if $n_\alpha \le n_\epsilon$, a circular compact sequence $C^n_{s,n_\epsilon - n_\alpha;\chi,\epsilon}$ with any starting position s can be achieved at the outputs of the RBN under a proper setting for switches in the network;*
2. *if $n_\alpha \ge n_\epsilon$, a circular compact sequence $C^n_{s,n_\alpha - n_\epsilon;\chi,\alpha}$ with any starting position s can be achieved at the outputs of the RBN under a proper setting for switches in the network.*

In the above theorem, we say that ϵ (or α) is the dominating type among ϵ and α if $n_\alpha \le n_\epsilon$ (or $n_\alpha \ge n_\epsilon$).

We now further explore the property of the circular compact sequence in order to deal with multicast assignments. In the case of merging two circular compact sequences with the same set of binary values, (for example, merging two sequences with χs and αs, $C^{\frac{n}{2}}_{s_0,l_0;\chi,\alpha}$ and $C^{\frac{n}{2}}_{s_1,l_1;\chi,\alpha}$), we can simply use the results in Theorem 1 or Lemma 1. However, in other cases, we may also need to merge two circular compact sequences with different sets of binary values (for example, merging a sequence with χs and αs, $C^{\frac{n}{2}}_{s_0,l_0;\chi,\alpha}$, and a sequence with χs and ϵs, $C^{\frac{n}{2}}_{s_1,l_1;\chi,\epsilon}$). In this case, we need to add two more switch settings: upper broadcast and lower broadcast, and use the notation of the trinary circular compact switch setting defined in Section 4.

To merge two compact sequences with different sets of binary values, the following question must be answered.

Question 2. *Given integers n, s, l, l_0, and l_1 satisfying that n is an even number, $0 \le s < n$, $0 \le l \le n$, $0 \le l_1 \le l_0 \le \frac{n}{2}$, and $l = l_0 - l_1$, do there exist integers s_0 and s_1 ($0 \le s_0, s_1 < \frac{n}{2}$) such that $C^{\frac{n}{2}}_{s_0,l_0;\chi,\alpha}$ and $C^{\frac{n}{2}}_{s_1,l_1;\chi,\epsilon}$ can be merged to $C^n_{s,l;\chi,\alpha}$ through an $n \times n$ merging network under a proper switch setting?*

The following lemma gives a positive answer to Question 2.

Lemma 2. *Given integers n, s, l, l_0, and l_1 satisfying that n is an even number, $0 \le s < n$, $0 \le l \le n$, $0 \le l_1 \le l_0 \le \frac{n}{2}$, and $l = l_0 - l_1$. Let*

$$s_0 = s \bmod \frac{n}{2},\ s_1 = (s + l) \bmod \frac{n}{2}$$

and the switch setting be

1. $W^{\frac{n}{2}}_{s_1,l_1;0,2}$, *if $s + l < \frac{n}{2}$;*
2. $W^{\frac{n}{2}}_{s_1,l_1,\frac{n}{2}-s_1-l_1;1,2,0}$, *if $s < \frac{n}{2}$ and $s + l \ge \frac{n}{2}$;*
3. $W^{\frac{n}{2}}_{s_1,l_1;1,2}$, *if $s \ge \frac{n}{2}$ and $s + l < n$;*

4. $W^{\frac{n}{2}}_{s_1,l_1,\frac{n}{2}-s_1-l_1;0,2,1}$, *if* $s \geq \frac{n}{2}$ *and* $s+l \geq n$.

Then, $C^{\frac{n}{2}}_{s_0,l_0;\chi,\alpha}$ *and* $C^{\frac{n}{2}}_{s_1,l_1;\chi,\epsilon}$ *can be merged to* $C^{n}_{s,l;\chi,\alpha}$ *through an* $n \times n$ *merging network under the above switch setting.*

Proof. See Appendix B. □

Question 2 has three other symmetric variants, which can be obtained by changing the condition in Lemma 2 to $l_1 \geq l_0$ and $l = l_1 - l_0$, and/or swapping α for ϵ. The corresponding solutions to these three variants are also symmetric. Lemmas 3, 4, and 5 give such solutions.

By swapping l_0 for l_1 in Lemma 2, we can obtain the following lemma.

Lemma 3. *Given integers* n, s, l, l_0, *and* l_1 *satisfying that* n *is an even number,* $0 \leq s < n$, $0 \leq l \leq n$, $0 \leq l_0 \leq l_1 \leq \frac{n}{2}$, *and* $l = l_1 - l_0$. *Let*

$$s_0 = (s+l) \bmod \frac{n}{2},\ s_1 = s \bmod \frac{n}{2}$$

and the switch setting be

1. $W^{\frac{n}{2}}_{s_0,l_0;1,2}$, *if* $s+l < \frac{n}{2}$;
2. $W^{\frac{n}{2}}_{s_0,l_0,\frac{n}{2}-s_0-l_0;0,2,1}$, *if* $s < \frac{n}{2}$ *and* $s+l \geq \frac{n}{2}$;
3. $W^{\frac{n}{2}}_{s_0,l_0;0,2}$, *if* $s \geq \frac{n}{2}$ *and* $s+l < n$;
4. $W^{\frac{n}{2}}_{s_0,l_0,\frac{n}{2}-s_0-l_0;1,2,0}$, *if* $s \geq \frac{n}{2}$ *and* $s+l \geq n$,

then, $C^{\frac{n}{2}}_{s_0,l_0;\chi,\alpha}$ *and* $C^{\frac{n}{2}}_{s_1,l_1;\chi,\epsilon}$ *can be merged to* $C^{n}_{s,l;\chi,\epsilon}$ *through an* $n \times n$ *merging network under the above switch setting.*

The two other variants are obtained by simply swapping α for ϵ, and changing upper broadcast to lower broadcast in Lemma 2 and Lemma 3, respectively.

Lemma 4. *Given integers* n, s, l, l_0, *and* l_1 *satisfying that* n *is an even number,* $0 \leq s < n$, $0 \leq l \leq n$, $0 \leq l_1 \leq l_0 \leq \frac{n}{2}$, *and* $l = l_0 - l_1$. *Let*

$$s_0 = s \bmod \frac{n}{2},\ s_1 = (s+l) \bmod \frac{n}{2}$$

and the switch setting be

1. $W^{\frac{n}{2}}_{s_1,l_1;0,3}$, *if* $s+l < \frac{n}{2}$;
2. $W^{\frac{n}{2}}_{s_1,l_1,\frac{n}{2}-s_1-l_1;1,3,0}$, *if* $s < \frac{n}{2}$ *and* $s+l \geq \frac{n}{2}$;
3. $W^{\frac{n}{2}}_{s_1,l_1;1,3}$, *if* $s \geq \frac{n}{2}$ *and* $s+l < n$;
4. $W^{\frac{n}{2}}_{s_1,l_1,\frac{n}{2}-s_1-l_1;0,3,1}$, *if* $s \geq \frac{n}{2}$ *and* $s+l \geq n$,

then, $C^{\frac{n}{2}}_{s_0,l_0;\chi,\epsilon}$ *and* $C^{\frac{n}{2}}_{s_1,l_1;\chi,\alpha}$ *can be merged to* $C^{n}_{s,l;\chi,\epsilon}$ *through an* $n \times n$ *merging network under the above switch setting.*

Lemma 5. *Given integers* n, s, l, l_0, *and* l_1 *satisfying that* n *is an even number,* $0 \leq s < n$, $0 \leq l \leq n$, $0 \leq l_0 \leq l_1 \leq \frac{n}{2}$, *and* $l = l_1 - l_0$. *Let*

$$s_0 = (s+l) \bmod \frac{n}{2},\ s_1 = s \bmod \frac{n}{2}$$

and the switch setting be

1. $W^{\frac{n}{2}}_{s_0,l_0;1,3}$, *if* $s+l < \frac{n}{2}$;
2. $W^{\frac{n}{2}}_{s_0,l_0,\frac{n}{2}-s_0-l_0;0,3,1}$, *if* $s < \frac{n}{2}$ *and* $s+l \geq \frac{n}{2}$;
3. $W^{\frac{n}{2}}_{s_0,l_0;0,3}$, *if* $s \geq \frac{n}{2}$ *and* $s+l < n$;
4. $W^{\frac{n}{2}}_{s_0,l_0,\frac{n}{2}-s_0-l_0;1,3,0}$, *if* $s \geq \frac{n}{2}$ *and* $s+l \geq n$,

then, $C^{\frac{n}{2}}_{s_0,l_0;\chi,\epsilon}$ *and* $C^{\frac{n}{2}}_{s_1,l_1;\chi,\alpha}$ *can be merged to* $C^{n}_{s,l;\chi,\alpha}$ *through an* $n \times n$ *merging network under the above switch setting.*

Now we are in the position to prove Theorem 3 by using Lemmas 1, 2 ,3, 4, and 5.

Proof of Theorem 3. By induction on the network size n. For $n = 2$ (base case), the network is a 2×2 switch. There are six possible cases:

- **Case 2.1.** $n_\alpha = n_\epsilon = 0$. Let the switch set to parallel. Then both outputs of the switch have χs, that is, sequence $C^2_{s,0;\chi,\alpha}$ (or equivalently $C^2_{s,0;\chi,\epsilon}$) can be achieved at the outputs of the switch for $0 \leq s \leq 1$.
- **Case 2.2.** $n_\alpha = n_\epsilon = 1$. Let the switch set to either upper-broadcast or lower-broadcast (depending on which input of the switch has α). Then both outputs of the switch have χs as in Case 2.1.
- **Case 2.3.** $n_\alpha = 1$ and $n_\epsilon = 0$. Let the switch set to parallel, if $s = 0$ and the upper input has α, or $s = 1$ and the lower input has α. Otherwise, let the switch set to crossing. Then sequence $C^2_{s,1;\chi,\alpha}$ can be achieved at the outputs of the switch for $0 \leq s \leq 1$.
- **Case 2.4.** $n_\alpha = 0$ and $n_\epsilon = 1$. Similar to Case 2.3.
- **Case 2.5.** $n_\alpha = 2$ and $n_\epsilon = 0$. Let the switch set to parallel. Then sequence $C^2_{s,2;\chi,\alpha}$ can be achieved at the outputs of the switch for $0 \leq s \leq 1$.
- **Case 2.6.** $n_\alpha = 0$ and $n_\epsilon = 2$. Similar to Case 2.5.

Hence, Theorem 3 holds for $n = 2$. Now, assume Theorem 3 holds for an $\frac{n}{2} \times \frac{n}{2}$ RBN (inductive hypothesis) and consider an $n \times n$ RBN. In the recursive definition of an $n \times n$ RBN, let n'_α and n'_ϵ denote the numbers of αs and ϵs, respectively in the upper $\frac{n}{2} \times \frac{n}{2}$ RBN, and n''_α and n''_ϵ denote the numbers of αs and ϵs in the lower $\frac{n}{2} \times \frac{n}{2}$ RBN, respectively. Clearly, we have

$$n'_\alpha + n''_\alpha = n_\alpha \text{ and } n'_\epsilon + n''_\epsilon = n_\epsilon.$$

We first assume $n_\alpha \leq n_\epsilon$ and consider the following cases:

- **Case $\frac{n}{2}$ 1.** $n'_\alpha \leq n'_\epsilon$ and $n''_\alpha \leq n''_\epsilon$. By the inductive hypothesis, Theorem 3 holds for both upper and lower $\frac{n}{2} \times \frac{n}{2}$ RBNs. That is, for any given integers s_0 and s_1 $(0 \leq s_0, s_1 < \frac{n}{2})$, $C^{\frac{n}{2}}_{s_0,n'_\epsilon-n'_\alpha;\chi,\epsilon}$ and $C^{\frac{n}{2}}_{s_1,n''_\epsilon-n''_\alpha\chi,\epsilon}$ can be achieved at the outputs of the upper and lower $\frac{n}{2} \times \frac{n}{2}$ RBNs, respectively. Now, let $l_0 = n'_\epsilon - n'_\alpha$, $l_1 = n''_\epsilon - n''_\alpha$, and $l = n_\epsilon - n_\alpha$, which implies $l = l_0 + l_1$. Then by Lemma 1, given any integer s $(0 \leq s < n)$, there exist integers s_0 and s_1 $(0 \leq s_0, s_1 < \frac{n}{2})$ such that $C^{\frac{n}{2}}_{s_0,l_0;\chi,\epsilon}$ and $C^{\frac{n}{2}}_{s_1,l_1;\chi,\epsilon}$ can be merged to $C^{n}_{s,l;\chi,\epsilon}$ through the $n \times n$ merging network under a proper switch setting. Hence, Theorem 3 holds in this case.
- **Case $\frac{n}{2}$ 2.** $n'_\alpha \leq n'_\epsilon$ and $n''_\alpha \geq n''_\epsilon$. By the inductive hypothesis, for any given integers s_0 and s_1 $(0 \leq s_0, s_1 < \frac{n}{2})$, $C^{\frac{n}{2}}_{s_0,n'_\epsilon-n'_\alpha;\chi,\epsilon}$ and $C^{\frac{n}{2}}_{s_1,n''_\alpha-n''_\epsilon;\chi,\alpha}$ can be achieved at the outputs of

the upper and lower $\frac{n}{2} \times \frac{n}{2}$ RBNs, respectively. Let $l_0 = n'_\epsilon - n'_\alpha$, $l_1 = n''_\alpha - n''_\epsilon$, and $l = n_\epsilon - n_\alpha$. Then we have $l = l_0 - l_1$ and $l_0 \geq l_1$. By Lemma 4, given any integer s $(0 \leq s < n)$, there exist integers s_0 and s_1 $(0 \leq s_0, s_1 < \frac{n}{2})$ such that $C^{\frac{n}{2}}_{s_0,l_0;\chi,\epsilon}$ and $C^{\frac{n}{2}}_{s_1,l_1;\chi,\alpha}$ can be merged to $C^{n}_{s,l;\chi,\epsilon}$ through the $n \times n$ merging network under a proper switch setting. Thus, the result is also true in this case.

- **Case $\frac{n}{2}$ 3.** $n'_\alpha \geq n'_\epsilon$ and $n''_\alpha \leq n''_\epsilon$. Similar to Case $\frac{n}{2}$ 2, by the inductive hypothesis and Lemma 3, we can see Theorem 3 holds in this case. □

Thus, we have proved Theorem 3 for $n_\alpha \leq n_\epsilon$. Symmetrically, we can show that the theorem holds for $n_\alpha \geq n_\epsilon$.

In the proof of Theorem 3, we can see that the number of the tag values of dominating type in the outputs of an RBN is the sum of those in its two sub-RBNs, if the two sub-RBNs have the same dominating type. In this case, Lemma 1 is applied, and we refer to it as ϵ/α-addition. On the other hand, the number of the tag values of dominating type is the difference between those of its two sub-RBNs, if the two-sub RBNs have different dominating types. In this case, Lemmas 2, 3, 4, and 5 are applied, and we refer to it as ϵ/α-elimination.

Finally, we are in the position to prove Theorem 2 by using Theorem 3.

Proof of Theorem 2. Since $n_\alpha \leq n_\epsilon$ holds for the original $n \times n$ RBN which is the scatter network of an $n \times n$ BSN (see (3)), by Theorem 3 we can always eliminate αs at the outputs of the RBN under a proper switch setting (i.e., $C^{n}_{s,n_\epsilon - n_\alpha;\chi,\epsilon}$ can be achieved at the outputs of the RBN, where s can be any number between 0 and $n-1$). On the other hand, from the proof of Theorem 3, we know that a tag value 0 or 1, once presented on a link at some stage of the RBN, will be passed in a unicast way to some output without any value change. We also know that value changes among 0s, 1s, αs, and ϵs occur only in some 2×2 switches. In this case, two inputs of the switch have α and ϵ, respectively, the switch setting is upper or lower broadcast (this is always true in our design), and two outputs of the switch have 0 and 1 respectively. Consequently, a pair of α and ϵ are transformed to a pair of 0 and 1. In total, n_α such pairs are transformed. Hence, $\widehat{n_0}$, $\widehat{n_1}$, $\widehat{n_\epsilon}$, and $\widehat{n_\alpha}$, which are the numbers of outputs with value 0, 1, ϵ, and α, respectively, satisfy the following:

$$\widehat{n_0} = n_0 + n_\alpha,\ \widehat{n_1} = n_1 + n_\alpha,\ \widehat{n_\epsilon} = n_\epsilon - n_\alpha,\ \text{and}\ \widehat{n_\alpha} = 0.$$

Also by using (1) and (2), we show that $0 \leq \widehat{n_0}, \widehat{n_1} \leq \frac{n}{2}$ and $\widehat{n_0} + \widehat{n_1} + \widehat{n_\epsilon} = n$. □

5.2 The Reverse Banyan Network as a Quasisorting Network

The results of Theorem 1 can be used to perform bit sorting in an RBN, that is, under a proper switch setting, all 0s and 1s on the inputs of the RBN can be routed to its outputs in an ascending order. However, this works only for full permutation assignments, in which each input of the RBN has a tag value either 0 or 1, and cannot be directly applied to partial permutation or multicast assignments.

TABLE 1
An Encoding Scheme for Tag Values

Tag	0	1	α	ϵ	ϵ_0	ϵ_1
$b_0b_1b_2$	000	001	100	$11X$	110	111

In the following, we discuss how an RBN can be used as a quasisorting network for partial permutation or multicast assignments. As can be seen in the last subsection, the outputs of the $n \times n$ RBN as a scatter network have tag values 0, 1, and ϵ only, which are passed to the inputs of the $n \times n$ RBN as a quasisorting network, and the numbers of such 0s or 1s are no more than $\frac{n}{2}$. The function of an $n \times n$ RBN as a quasisorting network is to route all 0s and 1s on the inputs of the RBN to the upper and lower halves of its outputs, respectively, and to route ϵs to the remaining positions at the outputs. We can let some of ϵs be dummy 0s (denoted as ϵ_0s) and the rest of ϵs be dummy 1s (denoted as ϵ_1s), such that both the number of all 0s (including all the real 0s and the dummy 0s) and the number of all 1s (including all the real 1s and the dummy 1s) are equal to $\frac{n}{2}$. Then by applying the bit sorting results in Theorem 1 we can achieve the quasisorting. The distributed algorithm dividing ϵs to ϵ_0s and ϵ_1s will be given in Section 6.

6 The Distributed Self-Routing Algorithms for RBNs

Lemmas 1, 2, 3, 4, and 5 actually provide a way to perform switch setting for all the switches in an RBN as a bit sorting network and as a scatter network, while switch setting for an RBN as a quasisorting network can be transformed to that for an RBN as a bit sorting network after all the ϵs on the inputs are properly divided into ϵ_0s and ϵ_1s.

In this section, we give a higher level description of the distributed self-routing algorithms used in switch setting for each type of RBNs: As a bit sorting network and as a scatter network, and describe a distributed algorithm for dividing ϵs to ϵ_0s and ϵ_1s in a quasisorting network. Needless to say, these switch setting algorithms can be implemented using proper logic circuits. In Section 7, we will discuss how they can be implemented in a pipelined fashion so that the circuits used for the switch setting can be distributed to each switch module and a lower hardware cost can be achieved.

By observing Lemmas 1, 2, 3, 4, and 5, we can see that switch setting at the last stage of RBN can be obtained if n, s, l_0, and l_1 are given. Before the switch setting being determined, l must be obtained from l_0 and l_1 (forward phase), and s_0 and s_1 must be obtained from s, l (or l_0 in Lemma 1) and n (backward phase). We can use the recursive property of an RBN to design a distributed routing algorithm for switch setting in the RBN.

Based on the recursive construction of an RBN, we can formulate the structure of an RBN into a complete binary tree shown in Fig. 8a. The root node of the tree represents the original RBN as a bit sorting network, a scatter network, or a quasisorting network; the two child nodes of the root represent the two recursively defined sub-RBN networks of

TABLE 2
Comparisons of Recursively Constructed Multicast Networks

Network	Cost	Depth	Routing time
Nassimi and Sahni's	$n \log^2 n$	$\log^2 n$	$\log^3 n$
Lee and Oruç's	$n \log^2 n$	$\log^2 n$	$\log^3 n$
New design	$n \log^2 n$	$\log^2 n$	$\log^2 n$
Feedback version	$n \log n$	$\log^2 n$	$\log^2 n$

the original RBN and so on; and, finally, the leaves of the tree represent the inputs of the original RBN.

The distributed algorithms are performed by each node of the binary tree. The algorithms start from leaves and perform forward computations all the way to the root, and then start from root and perform backward computations all the way to the leaves. For a clearer presentation, we omit the initialization step in each of the algorithms.

6.1 Distributed Switch Setting Algorithms

The switch setting at a node of the binary tree means the compact switch setting for all the switches in the last stage of the RBN represented by this node. Typically, there are three phases performed at each node of the tree. In the forward phase, whenever the two forward inputs l_0 and l_1 are available, l is computed and sent forward to the node in the next stage. In the backward phase, whenever the two forward inputs l_0 and l_1 and the backward input s are available, s_0 and s_1 are generated and sent backward to the nodes in the previous stage. In the switch setting phase, under the same precondition as the backward phase, each switch is set simultaneously by using the forward and backward input values and its own address in the stage.

The self-routing algorithm for the RBN as a bit sorting network is shown in Table 3 in Appendix C. It directly follows Lemma 1. The self-routing algorithm for the RBN as a scatter network is described in Table 4 in Appendix C. This algorithm actually combines the results of all cases in Lemmas 1, 2, 3, 4, and 5. The variable *type* in the algorithm represents the dominating type among αs and ϵs in the corresponding sub RBN network of size $n' \times n'$. If the two dominating types in the forward inputs agree (i.e., ϵ/α-addition), Lemma 1 applies as in the algorithm for the RBN as a bit sorting network; otherwise (i.e., ϵ/α-elimination) Lemmas 2, 3, 4, and 5 apply. The functions *BinaryCompactSetting*() and *TrinaryCompactSetting*() used in the algorithms correspond to the compact switch setting in those lemmas, and are described in Table 5 in Appendix C. All switches in one stage are set simultaneously according to the forward and backward values of the $n' \times n'$ subnetwork and its own switch-address in modulo $\frac{n'}{2}$.

6.2 Distributed ϵ-Dividing Algorithm in a Quasisorting Network

As stated in the last section, in an $n \times n$ RBN quasisorting network, the tag value on each input belongs only to

TABLE 3
The Distributed Self-Routing Algorithm for the RBN as a Bit Sorting Network

Forward phase:
  forward inputs: l_0, l_1.
  forward output: l.
  algorithm: {
    $l \leftarrow l_0 + l_1$;
  }

Backward phase:
  backward input: s.
  backward outputs: s_0, s_1.
  algorithm: {
    $s_0 \leftarrow s \bmod \frac{n'}{2}$;
    $s_1 \leftarrow (s + l_0) \bmod \frac{n'}{2}$;
  }

Switch setting phase:
  algorithm: {
    $b \leftarrow ((s + l_0)$ **div** $\frac{n'}{2}) \bmod \frac{n'}{2}$;
    BinaryCompactSetting(n', 0, s_1, $\bar{b}$, b);
  }

TABLE 4
The Distributed Self-Routing Algorithm for the RBN as a Scatter Network

Forward phase:

forward inputs: l_0, $type_0$, l_1, $type_1$.

forward outputs: l, $type$.

algorithm: {

if $(type_0 = type_1)$ { $type \leftarrow type_0$; $l \leftarrow l_0 + l_1$; }

else if $(l_0 \geq l_1)$ { $type \leftarrow type_0$; $l \leftarrow l_0 - l_1$; }

else /* $l_0 < l_1$ */ { $type \leftarrow type_1$; $l \leftarrow l_1 - l_0$;}

}

Backward phase:

backward input: s.

backward outputs: s_0, s_1.

algorithm: {

if $(type_0 = type_1)$ { $s_0 \leftarrow s \bmod \frac{n'}{2}$; $s_1 \leftarrow (s + l_0) \bmod \frac{n'}{2}$; }

else if $(l_0 \geq l_1)$ { $s_0 \leftarrow s \bmod \frac{n'}{2}$; $s_1 \leftarrow (s + l) \bmod \frac{n'}{2}$; }

else /* $l_0 < l_1$ */ { $s_0 \leftarrow (s + l) \bmod \frac{n'}{2}$; $s_1 \leftarrow s \bmod \frac{n'}{2}$;}

}

Switch setting phase:

algorithm: {

if $(type_0 = type_1)$ {$b \leftarrow ((s + l_0)$ **div** $\frac{n'}{2}) \bmod \frac{n'}{2}$;

BinaryCompactSetting(n', 0, s_1, $\bar{b}$, b); **return**;}

if $(type_0 = \alpha$ and $type_1 = \epsilon)$ { $bcast \leftarrow 2$; } /* upper broadcast */

else /* $type_0 = \epsilon$ and $type_1 = \alpha$ */ { $bcast \leftarrow 3$; } /* lower broadcast */

if $(l_0 \geq l_1)$ { $s_{tmp} \leftarrow s_1$; $l_{tmp} \leftarrow l_1$; $ucast \leftarrow 0$; } /* parallel setting */

else /* $l_0 < l_1$ */ { $s_{tmp} \leftarrow s_0$; $l_{tmp} \leftarrow l_0$; $ucast \leftarrow 1$; } /* crossing setting */

case $(s + l < \frac{n'}{2})$: BinaryCompactSetting(n', s_{tmp}, l_{tmp}, $ucast$, $bcast$);

case $(s < \frac{n'}{2}$ and $s + l \geq \frac{n'}{2})$: TrinaryCompactSetting(n', s_{tmp}, l_{tmp}, $\overline{ucast}$, $bcast$, $ucast$);

case $(s \geq \frac{n'}{2}$ and $s + l < n')$: BinaryCompactSetting(n', s_{tmp}, l_{tmp}, $\overline{ucast}$, $bcast$);

case $(s \geq \frac{n'}{2}$ and $s + l \geq n')$: TrinaryCompactSetting(n', s_{tmp}, l_{tmp}, $ucast$, $bcast$, $\overline{ucast}$);

}

$\{0, 1, \epsilon\}$, and the number of the inputs with tag 0 or the number of the inputs with tag 1 are no more than $\frac{n}{2}$. The function of ϵ-dividing algorithm is to reassign ϵ_0 (dummy 0) or ϵ_1 (dummy 1) to each input with tag value ϵ such that both the number of all 0s (including real 0s and dummy 0s) and the number of all 1s (including real 1s and dummy 1s) on the inputs of the network are equal to $\frac{n}{2}$.

The distributed ϵ-dividing algorithm is given in Table 6 in Appendix C. There are a forward phase and a backward phase performed at each node of the binary tree in Fig. 8a. The forward phase is to compute n_ϵ, the number of ϵs in the inputs of the sub-RBN represented by this node, when n'_ϵ and n''_ϵ, the numbers of ϵs in the inputs of its two sub-RBNs are available. Clearly, we have

$$n_\epsilon = n'_\epsilon + n''_\epsilon. \tag{6}$$

When the forward phase reaches the root node, the backward phase starts, which will determine that among n_ϵ of ϵs, how many should be dummy 0s (ϵ_0s) and how many should be dummy 1s (ϵ_1s). Denote the numbers as n_{ϵ_0} and n_{ϵ_1}, respectively. We must have

$$n_\epsilon = n_{\epsilon_0} + n_{\epsilon_1}, \tag{7}$$

and

$$n_{\epsilon_0} = n'_{\epsilon_0} + n''_{\epsilon_0}, \tag{8}$$

$$n_{\epsilon_1} = n'_{\epsilon_1} + n''_{\epsilon_1}, \tag{9}$$

TABLE 5
The Compact Switching Setting Algorithms for the Last Stage of an $n' \times n'$ RBN

```
  /* perform compact switch setting W^{n'/2}_{s,l;setting_1,setting_2} */
BinaryCompactSetting(n', s, l, setting_1, setting_2) {
    for (each switch in the last stage of the RBN) do in parallel {
        let i' be the address of the switch;
        i ← i' mod n'/2;
        case s + l < n'/2:
            if (s ≤ i < s + l) { r_{i'} ← setting_2; }
            else { r_{i'} ← setting_1; }
        case s + l ≥ n'/2:
            if (s + l − n'/2 ≤ i < s) { r_{i'} ← setting_1; }
            else { r_{i'} ← setting_2; }
    }
}
  /* perform compact switch setting W^{n'/2}_{s,l,n'/2−s−l;setting_1,setting_2,setting_3} */
TrinaryCompactSetting(n', s, l, setting_1, setting_2, setting_3) {
    for (each switch in the last stage of the RBN) do in parallel {
        let i' be the address of the switch;
        i ← i' mod n'/2;
        case 0 ≤ i < s:          r_{i'} ← setting_1;
        case s ≤ i < s + l:      r_{i'} ← setting_2;
        case s + l < n'/2:       r_{i'} ← setting_3;
}
```

where n'_{ϵ_0} and n'_{ϵ_1}, and n''_{ϵ_0} and n''_{ϵ_1} are the numbers of dummy 0s and dummy 1s for two child nodes, respectively. Equations (6), (8), and (9) are invariants for all but leaf nodes and (7) is an invariant for all nodes. To balance the number of 0s and the number of 1s (both "real" and "dummy" ones), initially in the backward phase, n_{ϵ_0} and n_{ϵ_1} of the root node can be set to

$$n_{\epsilon_1} = \frac{n}{2} - n_1 \text{ and } n_{\epsilon_0} = n_\epsilon - n_{\epsilon_1}$$

where n_1 represents the number of the real 1s of the RBN, which can also be computed through the forward phase. Then, n'_{ϵ_0} and n'_{ϵ_1}, and n''_{ϵ_0} and n''_{ϵ_1} of the child node can be computed and passed backward as in the backward algorithm. That is, when forward inputs n'_ϵ and n''_ϵ and backward inputs n_{ϵ_0} and n_{ϵ_1} are available, the backward outputs is computed as follows:

$$n'_{\epsilon_0} = \min\{n_{\epsilon_0}, n'_\epsilon\},\ n'_{\epsilon_1} = n'_\epsilon - n'_{\epsilon_0},$$
$$n''_{\epsilon_0} = n_{\epsilon_0} - n'_{\epsilon_0},\text{ and } n''_{\epsilon_1} = n''_\epsilon - n'_{\epsilon_1}.$$

It can be verified that the invariants shown in (6), (7), (8), and (9) hold at every node. Finally, in the last step of the algorithm, if the n_{ϵ_0} of a leaf node is 1, this input is assigned an ϵ_0, and if the n_{ϵ_1} of the leaf node is 1, this input is assigned an ϵ_1.

7 Implementation Issues and Complexity Analyses

In this section, we first discuss some implementation issues pertaining to the newly designed self-routing multicast network, and then analyze the complexity of the network.

7.1 Routing Tag Format and Handling

To send a multicast message (i.e., multidestination message) through a multicast network, the header of the message must carry the information of the addresses of the destination nodes. This information, as an overhead due to the nature of this type of communication, is considered as part of the message data. The preparation of the header is done before the message enters the network. For more detailed discussions on this issue and various multiaddress encoding schemes, readers can refer to [3]. In this subsection, we introduce a multiaddress encoding scheme for the routing tag of the proposed self-routing multicast network, and briefly describe the routing tag handling technique.

TABLE 6
The Distributed Algorithm Dividing ϵs to ϵ_0s and ϵ_1s in the RBN as a Quasi-Sorting Network

Forward phase:	**Backward phase:**
forward inputs: n'_ϵ, n''_ϵ.	**backward input:** n_{ϵ_0}, n_{ϵ_1}.
forward output: n_ϵ.	**backward outputs:** n'_{ϵ_0}, n'_{ϵ_1} n''_{ϵ_0}, n''_{ϵ_1}.
algorithm: {	**algorithm:** {
$n_\epsilon \leftarrow n'_\epsilon + n''_\epsilon$;	$n'_{\epsilon_0} \leftarrow \min\{n_{\epsilon_0}, n'_\epsilon\}$; $n'_{\epsilon_1} \leftarrow n'_\epsilon - n'_{\epsilon_0}$;
}	$n''_{\epsilon_0} \leftarrow n_{\epsilon_0} - n'_{\epsilon_0}$; $n''_{\epsilon_1} \leftarrow n''_\epsilon - n'_{\epsilon_1}$;
	}

In order to describe the multicast routing tag format, we first need the following preparations.

A multicast, represented by a set of addresses of its destinations in binary, in an $n \times n$ BRSMN can be expressed as a complete binary tree of height $\log n$ (number of levels) with each node assigned a tag value chosen from $\{0, 1, \alpha, \epsilon\}$. In this binary tree, level i $(1 \le i \le \log n)$ gives the information of the ith most significant bits of the destinations of the multicast. The binary tree can also be defined recursively as follows: The root node (i.e., the unique node in level 1) of the tree represents the entire multicast, and its left (or right) child node in level 2, represents a multicast which is formed by all the destination addresses of the original multicast with the most significant bit being 0 (or 1). In general, for a node in level i $(1 \le i \le \log n - 1)$, its left (or right) child node in level $i+1$ represents a multicast which is formed by all the destination addresses of the multicast at this node in level i with the ith most significant bit being 0 (or 1). The tag assigned to each node in the binary tree depends on the multicast it represents. For a node in level i $(1 \le i \le \log n)$, it is assigned the tag α if and only if the destination addresses of the multicast it represents have both 0 and 1 in the $i$the most significant bit; it is assigned the tag 0 (or 1) if, and only if, the destination addresses of the multicast it represents have only 0 (or 1) in the ith most significant bit; it is assigned the tag ϵ if and only if it represents an empty multicast. Clearly, for a node in level i $(1 \le i \le \log n - 1)$, if it has a tag α, both its children must have a tag other than ϵ; if it has a tag 0 (or 1), its left (or right) child must have a tag other than ϵ, and its right (or left) child must have a tag ϵ; and if it has a tag ϵ, both its children have a tag ϵ. It can be shown that, for a given multicast, the binary tree with tags defined above is unique. Fig. 9a and Fig. 9b give two examples of the binary trees with a tag at each node and their corresponding multicasts.

Now we are in the position to describe the multicast routing tag format and its handling technique. First of all, any network input without a message is always assumed to have a tag ϵ. Each message at a network input is assigned a routing tag sequence $a_0, a_1, a_2, \ldots, a_{2n-3}, a_{2n-2}$, which corresponds to the complete binary tree of the multicast on the input. We use a_0 as the routing tag to route the message on this input in the $n \times n$ BSN, i.e., set up switches in the scatter and quasisorting networks as described in Section 5; then pass the rest of tags along the setup path(s) through the $n \times n$ BSN so that the tags can be used as routing tag sequence(s) of the upper or lower or both $\frac{n}{2} \times \frac{n}{2}$ BSNs. As shown in Fig. 10, if a_0 equals α, it passes $a_1, a_3, \ldots, a_{2n-3}$ to the upper $\frac{n}{2} \times \frac{n}{2}$ BSN and $a_2, a_4, \ldots, a_{2n-2}$ to the lower $\frac{n}{2} \times \frac{n}{2}$ BSN simultaneously; if a_0 equals 0 (or 1), it passes only $a_1, a_3, \ldots, a_{2n-3}$ (or $a_2, a_4, \ldots, a_{2n-2}$) to the upper (or lower) $\frac{n}{2} \times \frac{n}{2}$ BSN. Passing a_is alternatively to the upper and the lower subnetworks is for efficiency consideration. In fact, this way only a constant number of buffers are needed to store the tag sequence at each input of a BSN as it passes through the network. To ensure the message is sent to the destinations of the multicast, the sequences $a_1, a_3, \ldots, a_{2n-3}$ and $a_2, a_4, \ldots, a_{2n-2}$ must match the left and the right subtrees of the original complete binary tree. This should also hold for all smaller BSNs. Thus, the routing tag sequence $a_0, a_1, a_2, \ldots, a_{2n-3}, a_{2n-2}$ consists of all tags of the complete binary tree arranged in a proper order. In the following, we give a formal definition of the routing tag sequence.

Let SEQ denote the final routing tag sequence for a multicast. Let the tags of all 2^{i-1} nodes (from left to right) in level i $(1 \le i \le \log n)$ be $t_{i1}, t_{i2}, \ldots, t_{i,2^{i-2}}, t_{i,2^{i-2}+1}, \ldots, t_{i,2^{i-1}}$ which is denoted as sequence SEQ_i (Fig. 11 gives an example for $n = 16$). Let *conc* be a function which concatenates a number of sequences to form a single sequence, *merge* be a function which merges two sequences of equal length and is defined as:

$$merge(b_1, b_2, \ldots, b_k; c_1, c_2, \ldots, c_k) = b_1, c_1, b_2, c_2, \ldots, b_k, c_k \tag{10}$$

and *order* be a recursive function of a sequence of length $k = 2^i$ which is defined as:

$$\begin{aligned} order(b_1, \ldots, b_k) = merge(&order(b_1, \ldots, b_{\frac{k}{2}}),\\ &order(b_{\frac{k}{2}+1}, \ldots, b_k)), \text{ and } order(b_1) = b_1. \end{aligned} \tag{11}$$

Finally, the routing tag sequence is defined as:

$$\begin{aligned} SEQ = conc(&order(SEQ_1), order(SEQ_2), \ldots,\\ &order(SEQ_{\log n})). \end{aligned} \tag{12}$$

In the example shown in Fig. 11 for $n = 16$, we have the ordered sequences of four levels,

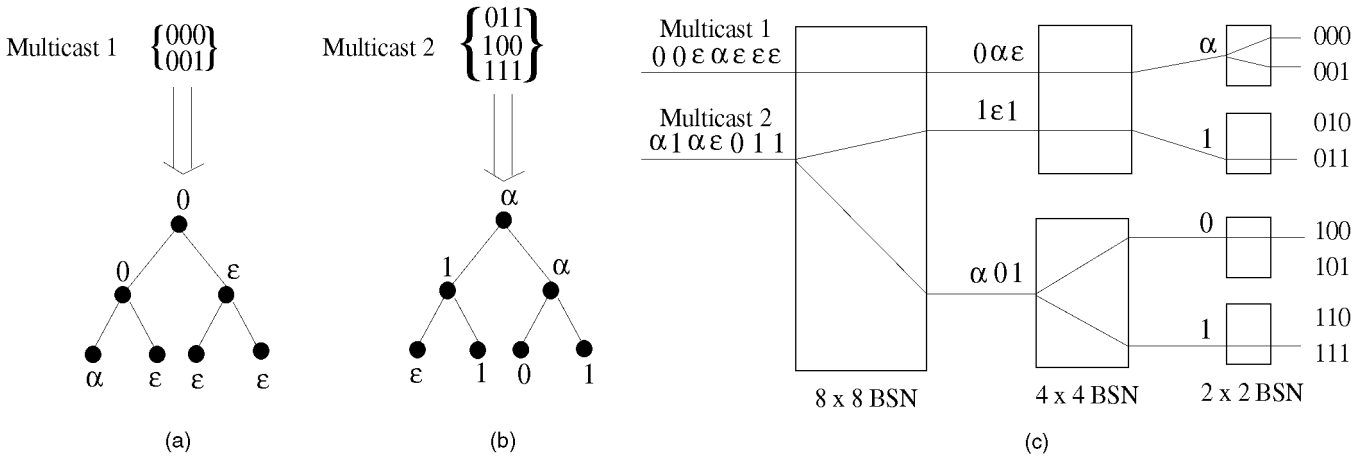

Fig. 9. (a) and (b) Examples of the binary trees with tags and their corresponding multicasts. (c) Routing tag sequences for (a) and (b) and their routing.

$$order(SEQ_1) = t_{11},$$
$$order(SEQ_2) = t_{21}, t_{22},$$
$$order(SEQ_3) = t_{31}, t_{33}, t_{32}, t_{34}, \text{ and}$$
$$order(SEQ_4) = t_{41}, t_{45}, t_{43}, t_{47}, t_{42}, t_{46}, t_{44}, t_{48}.$$

Thus, the final routing tag format is their concatenation:

$$t_{11}, t_{21}, t_{22}, t_{31}, t_{33}, t_{32}, t_{34}, t_{41}, t_{45}, t_{43}, t_{47}, t_{42}, t_{46}, t_{44}, t_{48}. \quad (13)$$

We omit the proof of (12). Instead, we verify its correctness by checking the tag sequence (13) in the example and, for simplicity, we check only one branch of the binary tree. By using our tag handling method, the tag sequence $t_{21}, t_{31}, t_{32}, t_{41}, t_{43}, t_{42}, t_{44}$, which corresponds to the left subtree, is sent to the upper 8×8 BSN; and then the tag sequence t_{31}, t_{41}, t_{42}, which corresponds to the first subtree in level 3, is sent to the upper 4×4 BSN, and so on. For the multicast examples in Fig. 9a and Fig. 9b, their routing tag sequences are $00\epsilon\alpha\epsilon\epsilon\epsilon$ and $\alpha 1\alpha\epsilon 011$, respectively, and Fig. 9c shows the handling for these two sequences.

7.2 Circuit Design Issues

Based on the distributed self-routing algorithms presented in the last section, we can design a self-routing circuit for our RBN-based multicast network in a similar approach to Cheng and Chen's [14] self-routing circuit design for their RBN-based permutation network. In this subsection, we only briefly discuss some circuit design issues.

First of all, as can be seen, the different tag values 0, 1, α, and ϵ (including ϵ_0 and ϵ_1) are used in the RBN as a scatter network and as a quasisorting networks. We need to use three bits, $b_0 b_1 b_2$ to represent a tag value. Table 1 lists an encoding scheme for these tag values. Also, when we compute the numbers of αs, ϵs, and 1s on the inputs, we need a combination of all the bits for every input tag. In this encoding, we can use $b_0 \wedge \bar{b_1}$ of an input tag for counting the number of α for this input, and use $b_0 \wedge b_1$ for counting the number of ϵ of an input in the forward phases of self-routing for the RBN as a scatter network and ϵ-dividing algorithm for the RBN as a quasisorting network. Furthermore, since only 0s, 1s, and ϵs (including ϵ_0s and ϵ_1s) can be the input tags for the RBN as a quasi-sorting network, we need to use only bit b_2 of an input tag for counting all 1s (including real and dummy 1s) in the forward phase of self-routing for this RBN.

Second, the binary tree structure used in all distributed algorithms can be embedded into an RBN. For a balanced hardware distribution, we separate the tree to a forward tree and a backward tree as shown in Fig. 8b, where the first and the last switches of the last stage of a sub-RBN network can serve as the nodes in the forward tree and the backward tree, respectively, and the switches in between can use the results from these two nodes for their switch settings.

Third, we can see that the most frequently used operation in the distributed algorithms is addition (or addition-like operations). For an $n \times n$ RBN, the maximum

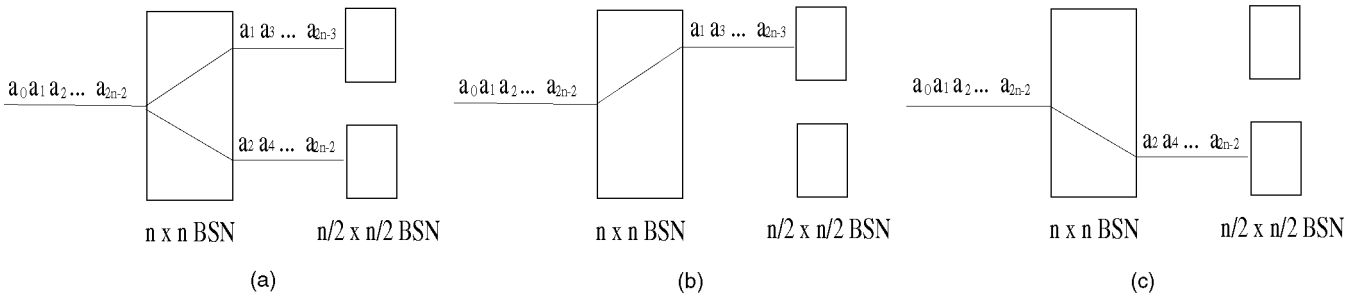

Fig. 10. Three possible cases of routing in an $n \times n$ BSN.

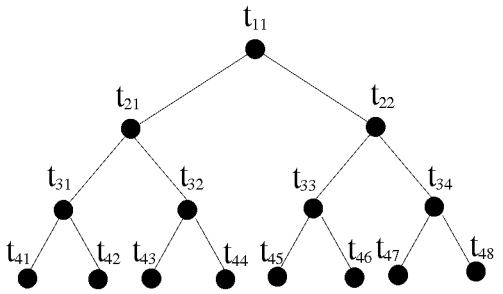

Fig. 11. A binary tree represents the routing tag sequence of a multicast for $n = 16$.

values of n_α, n_ϵ, or n_1 are n, which can be represented by using at most $\log n$ bits. We implement the adder in a pipelined fashion as shown in Fig. 12. Then, the $\log n$ bit adder can be reduced to a one bit adder. Also, since the distributed algorithms work in a pipelined fashion, the delay caused by running the forward and the backward processes is also reduced.

7.3 Feedback Implementation of the Network

Note that in our design all major functional components are recursively defined reverse banyan networks. We can easily reuse part of the network to reduce network cost. In particular, we can construct a feedback version of our multicast network, BRSMN as depicted in Fig. 13. Let an $n \times n$ BSN use only one $n \times n$ RBN, with each output fed back to the input with the same address. The first pass on the RBN functions as a scatter network and the second pass functions as a quasisorting network. Note that in the original design of the BRSMN, (see Fig. 2 for an example), the $n \times n$ BSN is followed by two $\frac{n}{2} \times \frac{n}{2}$ BSNs, and so on. Now, a BSN is simply an RBN of the same size and an $n \times n$ RBN consists of two $\frac{n}{2} \times \frac{n}{2}$ RBNs followed by an $n \times n$ merging network (see Fig. 5). In the feedback version of the network, after the first split according to the most significant bit of a destination address, we can reuse the two $\frac{n}{2} \times \frac{n}{2}$ RBNs in the $n \times n$ RBN as the two BSNs to perform the second split according to the second most significant bit of the destination address, and so on. Consequently, the feedback version of an $n \times n$ BRSMN is simply an $n \times n$ RBN.

7.4 Complexity Analyses

In this subsection, we analyze the hardware cost, network depth and routing time of the new multicast network. Compared to Chen and Cheng's permutation network [14], since we add only a constant cost (i.e., a constant number of one bit adders or adder-like circuits) to each switch for the self-routing circuit and there are $\frac{n}{2}\log n$ switches in an $n \times n$ RBN, the hardware cost of the RBN is $O(n \log n)$, and so is that of an $n \times n$ BSN. Let $C(n)$ denote the cost of an $n \times n$ BRSMN. By the recursive construction in Fig. 1, we have $C(n) = O(n \log n) + 2C(\frac{n}{2})$, which implies $C(n) = O(n \log^2 n)$.

Since the network depth of an $n \times n$ RBN is $O(\log n)$, so is an $n \times n$ BSN. Let $D(n)$ denote the network depth for an $n \times n$ BRSMN. From $D(n) = O(\log n) + D(\frac{n}{2})$, we immediately have $D(n) = O(\log^2 n)$.

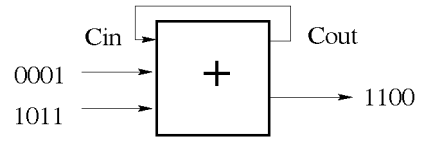

Fig. 12. One bit adder used in a pipelined fashion.

Note that the distributed routing algorithms work in a pipelined fashion. It takes $O(\log n)$ unit time delay for the first bit (from input) to reach a switch in the last stage of an $n \times n$ RBN. Then it takes only $O(1)$ unit time for each of the subsequent $\log n - 1$ bits to reach a switch in the last stage. Hence, the propagation delay in the forward phase is $O(\log n)$ unit time, and so is in the backward phase. Let $T(n)$ denote the total propagation delay of the routing algorithm for an $n \times n$ BRSMN. Since there are a constant number of forward and backward phases in switch setting in a BSN, the routing time for an $n \times n$ BSN is $O(\log n)$. From $T(n) = O(\log n) + T(\frac{n}{2})$, we obtain $T(n) = O(\log^2 n)$.

For the feedback implementation discussed in Section 7.3, since an $n \times n$ BRSMN is now simply an $n \times n$ RBN and the routing circuit distributed in each switch still has $O(1)$ cost, we conclude that the feedback version of our design has $O(n \log n)$ network cost.

In Table 2, we list the cost, depth and routing time of the newly designed multicast network along with those of the existing multicast networks constructed by the recursive decomposition approach. Finally, we can see that the RBN-based multicast network presented in this paper achieves the same order of complexities of network cost, network depth, routing time as those of the RBN-based permutation network proposed by Cheng and Chen [14].

8 Conclusions

In this paper, we have proposed a design for a new self-routing multicast network using an approach based on recursive decompositions of multicast networks. Different from earlier proposed multicast networks, the

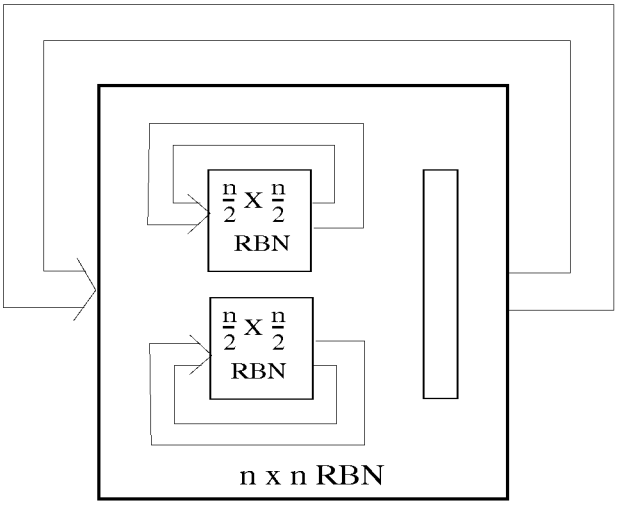

Fig. 13. The feedback implementation of the network.

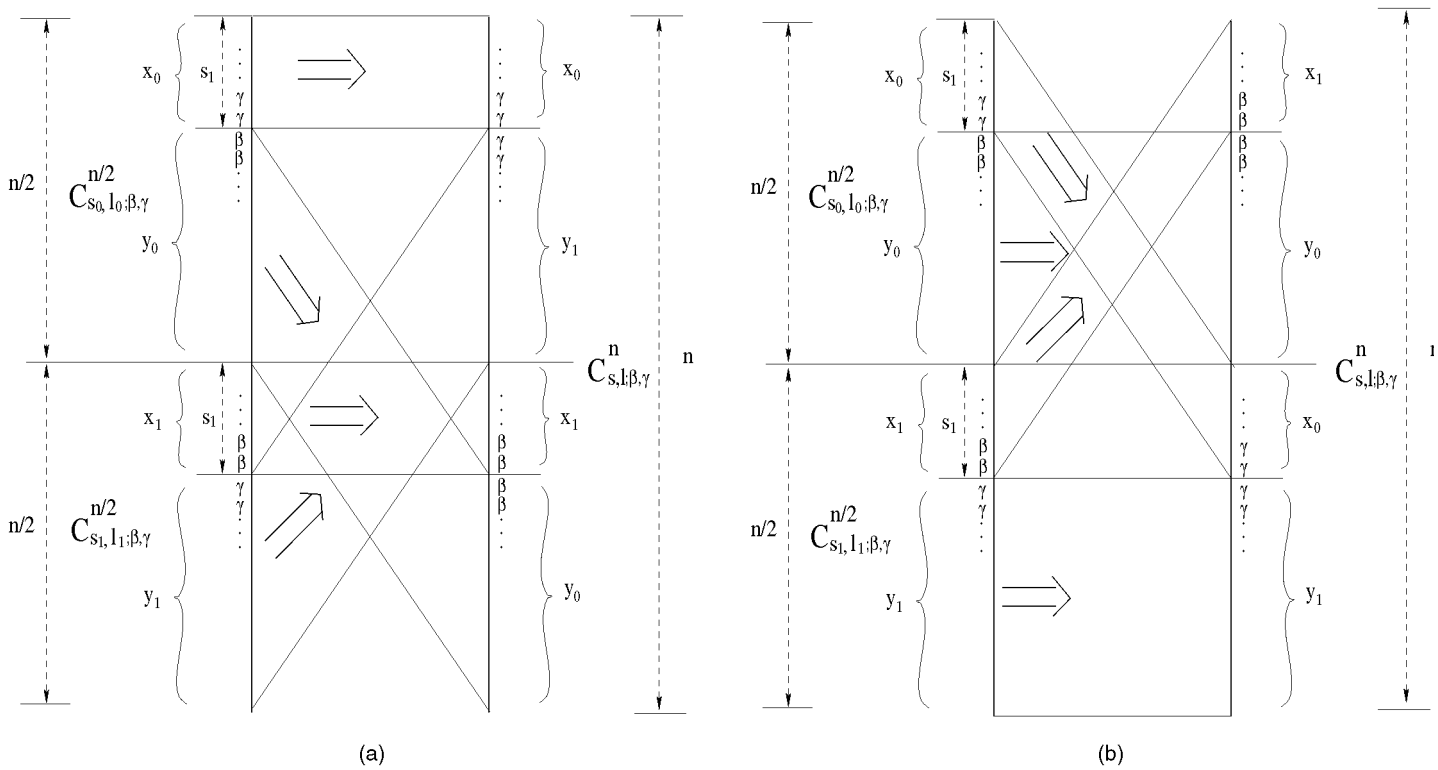

Fig. 14. Illustration of the proof of Lemma 14.

new multicast network is conceptually simple and has good modularity. The network design is based on the binary radix sorting concept and all functional components of the network are recursively constructed reverse banyan networks. Thus, it has a potential to greatly reduce the network cost by reusing part of the network. In addition, the routing circuit is completely distributed into each switch of the network so that the network is operated in a self-routing manner. The new multicast network we design compares favorably with the previously proposed multicast networks. It uses $O(n \log^2 n)$ logic gates, and has $O(\log^2 n)$ depth and $O(\log^2 n)$ routing time where the unit of time is a gate delay. By reusing part of the network, the feedback version of our design can further reduce the network cost to $O(n \log n)$.

APPENDIX A

Proof of Lemma 1. In this appendix, we prove that two $\frac{n}{2}$-bit circular compact sequences $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ and $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ can be merged to $C^{n}_{s,l;\beta,\gamma}$ through an $n \times n$ merging network by the given switch setting.

In Fig. 14a (or Fig. 14b), on the left side both $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ and $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ are expressed as vertical segments of length $\frac{n}{2}$, and on the right side $C^{n}_{s,l;\beta,\gamma}$ is expressed as a vertical segment of length n. Note that the first s_1 consecutive switches all are set to setting b, and the rest of $\frac{n}{2} - s_1$ consecutive switches are set to the opposite setting $\bar{b}$. Thus, the segment for $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ can be divided into two subsegments x_0 (of length s_1) and y_0 (of length $\frac{n}{2} - s_1$). Similarly, the segment for $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ can be divided into two subsegments x_1 (of length s_1) and y_1 (of length $\frac{n}{2} - s_1$). Note that the segment $x_1 \circ y_1$ (where symbol $\circ$ denotes a concatenation operation) represents $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ and x_1 is of length s_1 which equals the starting position of the γs sequence in $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$. We must have that sequence x_1 ends with β and sequence y_1 starts with γ. Also, since $s_1 = (s + l_0) \bmod \frac{n}{2} = (s_0 + l_0) \bmod \frac{n}{2}$ and x_0 is of length s_1, in segment $x_0 \circ y_0$ we must have that sequence x_0 ends with γ and sequence y_0 starts with β. We consider two cases of b values.

Case 1. $b = 0$ (Fig. 14a). All inputs in both subsegments x_0 and x_1 on the left are routed to the outputs on the right in a parallel way, and all inputs in both subsegments y_0 and y_1 on the left are routed to the outputs on the right in a crossing way. Therefore, the segment on the right is $x_0 \circ y_1 \circ x_1 \circ y_0$. We now prove that the sequence represented by the segment on the right is indeed the circular compact sequence $C^{n}_{s,l;\beta,\gamma}$. In the segment $x_0 \circ y_1 \circ x_1 \circ y_0$, x_0 ending with γ and y_1 starting with γ make the γs in $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ and $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ consecutive at the joint of x_0 and y_1. Also, x_1 ending with β and y_0 starting with β make the βs in $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ and $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ consecutive at the joint of x_1 and y_0. Moreover, the consecutiveness of β or γ is preserved at the joint of y_1 and x_1, since $x_1 \circ y_1$ is a circular compact sequence. A similar argument applies to the joint of y_0 and x_0. Thus, the segment $x_0 \circ y_1 \circ x_1 \circ y_0$ is a circular compact sequence since l_0 γs from $C^{\frac{n}{2}}_{s_0,l_0;\beta,\gamma}$ and l_1 γs from $C^{\frac{n}{2}}_{s_1,l_1;\beta,\gamma}$ are concatenated in the segment. Finally, we can check that the starting position for γs sequence in $x_0 \circ y_1 \circ x_1 \circ y_0$ is s. Thus, the merged sequence is $C^{n}_{s,l;\beta,\gamma}$. Note that $b = 0$ implies $\left[(s + l_0) \textbf{ div } \frac{n}{2}\right] \bmod 2 = 0$. It follows that in this case we must have either $s + l_0 < \frac{n}{2}$

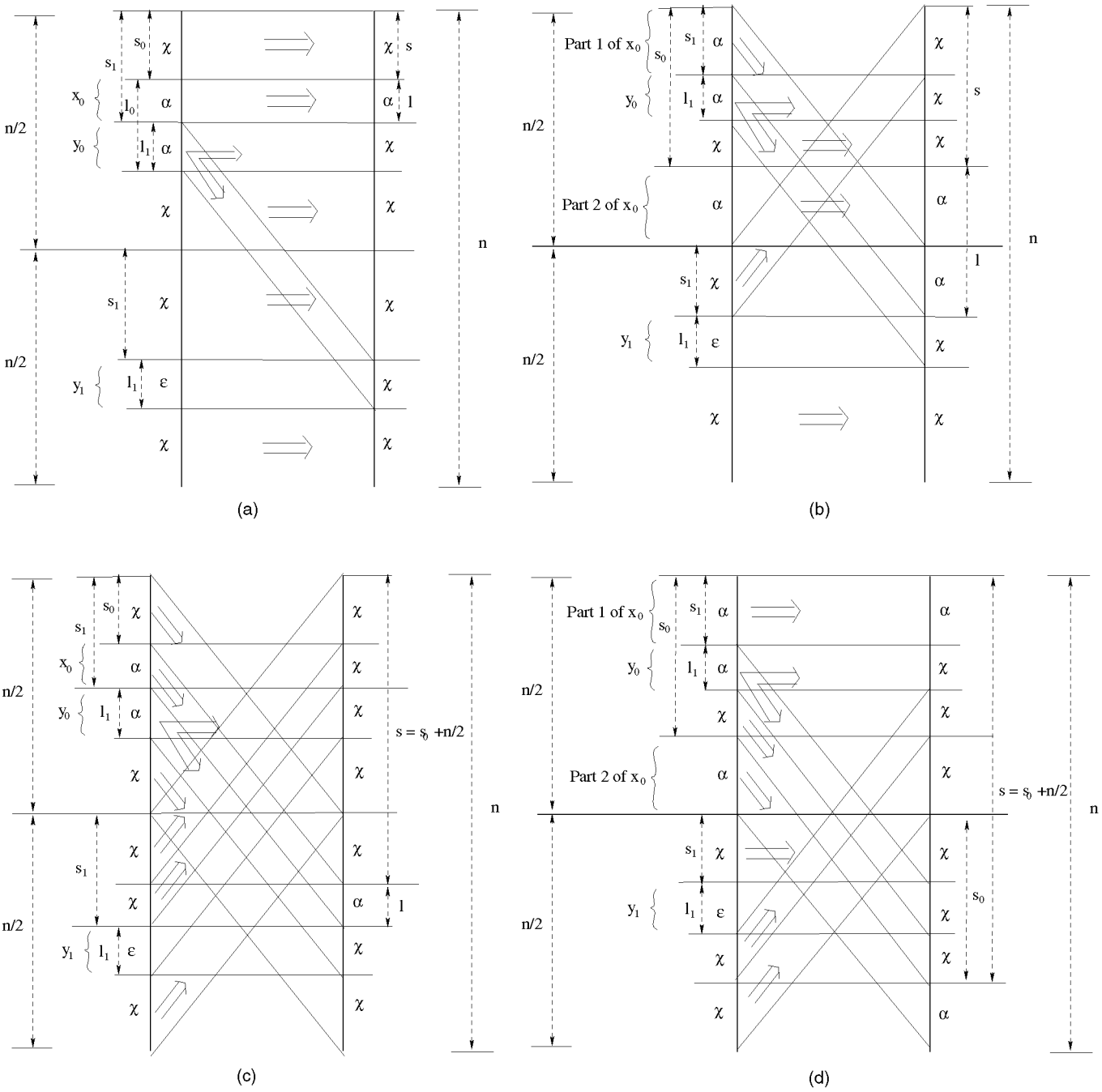

Fig. 15. Illustration of the proof of Lemma 2.

or $s + l_0 \geq n$ (of course, $s + l_0 < \frac{3n}{2}$). Also, from $s_1 = (s + l_0) \bmod \frac{n}{2}$, we have $s = (s_1 - l_0) \bmod n$. In the sequence $x_0 \circ y_1 \circ x_1 \circ y_0$, the ending point (i.e., bottom) of x_0 is at position s_1. From this point going upward (in a circular way), there are consecutive l_0 γs, and the last γ is at the starting position for γs sequence in $x_0 \circ y_1 \circ x_1 \circ y_0$, which is $(s_1 - l_0) \bmod n$ in this case.

Case 2. $b = 1$ (Fig. 14b). Similar to Case 1, all inputs in both subsegments x_0 and x_1 on the left are routed to the outputs on the right in a crossing way, and all inputs in both subsegments y_0 and y_1 on the left are routed to the outputs on the right in a parallel way. The segment on the right now becomes $x_1 \circ y_0 \circ x_0 \circ y_1$. Similarly, it can be verified that the merged sequence is a circular compact sequence with $l = l_0 + l_1$ consecutive γs and $n - l$ consecutive βs in a circular way. The remaining is to prove that it is with starting point s. Note that $b = 1$ implies $\left[(s + l_0) \textbf{ div } \frac{n}{2}\right] \bmod 2 = 1$. It follows, therefore, that $\frac{n}{2} \leq s + l_0 < n$. We can obtain $s = s_1 - l_0 + \frac{n}{2}$. Now, in sequence $x_1 \circ y_0 \circ x_0 \circ y_1$, the ending point (i.e., bottom) of x_0 is at position $s_1 + \frac{n}{2}$. From this point going upward, there are consecutive l_0 γs, and then the last γ is at the starting position for γs sequence in $x_1 \circ y_0 \circ x_0 \circ y_1$, which is $s_1 + \frac{n}{2} - l_0$ in this case. □

APPENDIX B

Proof of Lemma 2. In this appendix, we show the correctness of Lemma 2. First, it should be pointed out that since $0 \leq l \leq l_0$, the four cases in Lemma 2 (i.e., 1) $s + l < \frac{n}{2}$, 2) $s < \frac{n}{2}$ and $s + l \geq \frac{n}{2}$, 3) $s \geq \frac{n}{2}$ and $s + l < n$, and 4) $s \geq \frac{n}{2}$ and $s + l \geq n$) cover all possible intervals. Applying a similar approach to the proof of Lemma 1, we represent the input and output sequences as vertical segments as shown in Fig. 15a, Fig. 15b, Fig. 15c, and Fig. 15d, with one subfigure for each of the four cases.

Note that all αs on the inputs are in the upper half and all ϵs on the inputs are in the lower half. Now, let's first look at the two subsegments y_0 and y_1 on the left in Fig. 15a. In the lower half of the inputs (i.e., $C^{\frac{n}{2}}_{s_1,l_1;\chi,\epsilon}$), the subsegment y_1 which starts from position s_1 and is of length l_1 (in a circular way, modulo $\frac{n}{2}$) consists of consecutive ϵs. Also, since

$$s_1 = (s + l) \bmod \frac{n}{2} = (s_0 + l) \bmod \frac{n}{2} = (s_0 + l_0 - l_1) \bmod \frac{n}{2},$$

in the upper half of the inputs (i.e., $C^{\frac{n}{2}}_{s_0,l_0;\chi,\alpha}$), the subsegment y_0 which starts from position s_1 and is of length l_1 consists of consecutive αs. Thus, these two subsegments from the upper and lower halves of the inputs are at the same position within their segment and have a distance of $\frac{n}{2}$. A similar observation can be drawn for Fig. 15b, Fig. 15c, and Fig. 15d. In addition, in the switch settings (1), (2), (3), and (4) in Lemma 2, only those l_1 consecutive switches (in a circular way) starting from the s_1^{th} switch have an upper broadcast setting. That is, all corresponding αs and ϵs in these two subsegments y_0 and y_1 are neutralized (or eliminated) and become χs in the corresponding positions on the outputs of the RBN.

The rest of the consecutive αs on the inputs form a subsegment named x_0 as shown in Fig. 15a, Fig. 15b, Fig. 15c, and Fig. 15d. x_0 starts from position s_0, ends at position $s_1 - 1$ (in a circular way, modulo $\frac{n}{2}$) in the upper half, and is of length $l = l_0 - l_1$. Next, we will show how the αs in x_0 are mapped to the predetermined positions in the outputs of the RBN case by case.

Case 1. $s + l < \frac{n}{2}$ (Fig. 15a). In this case, since $s + l < \frac{n}{2}$, all l αs on the outputs will be in the upper half. Note that in switch setting $W^{\frac{n}{2}}_{s_1,l_1;0,2}$, all the switches except those with an upper broadcast setting have a parallel setting. Since the subsegment x_0 (consisting of all αs) in the upper half of the inputs is mapped to the outputs in a parallel way, the l consecutive outputs starting from $s = s_0$ have αs, and the rest of the outputs have χs. Hence, the entire sequence of the outputs is $C^n_{s,l;\chi,\alpha}$.

Case 2. $s < \frac{n}{2}$ and $s + l \geq \frac{n}{2}$ (Fig. 15b). In this case, the segment of the consecutive outputs with αs will go across the middle point of the entire segment of all n outputs. However, all αs on the left are in the upper half of the inputs. This indicates that we need to map some αs on the inputs to the outputs in a parallel way and map some αs in a crossing way. The switch setting $W^{\frac{n}{2}}_{s_1,l_1,\frac{n}{2}-s_1-l_1;1,2,0}$ can accomplish this. We can see that the subsegment x_0 (consisting of all αs) in the upper half of the inputs can be divided into two parts. Since $s_0 = s$ and $s_1 = s + l - \frac{n}{2}$, the sub-segment of the inputs with αs starting at $s_0 = s$ and ending at $\frac{n}{2} - 1$ is mapped to the outputs in a parallel way, and the subsegment of the inputs with αs starting at 0 and ending at $s_1 - 1$ is mapped to the outputs in a crossing way, that is, mapped to a subsegment of the outputs starting at $\frac{n}{2}$ and ending at $\frac{n}{2} + s_1 - 1$. In other words, we obtain that $l = s_1 + \frac{n}{2} - s$ consecutive outputs have αs starting at $s = s_0$, and the rest of the outputs have χs, which is $C^n_{s,l;\chi,\alpha}$.

Case 3. $s \geq \frac{n}{2}$ and $s + l < n$ (Fig.15c). In this case, all l αs on the outputs will be in the lower half part. Similar to Case 1, since $s = s_0 + \frac{n}{2}$ and $s_1 = s_0 + l$, the subsegment x_0 starting at s_0 and of length l in the upper half of the inputs is mapped to the outputs in a crossing way. Thus, the l consecutive outputs staring from $s = s_0 + \frac{n}{2}$ have αs, and the rest of the outputs have χs, which is $C^n_{s,l;\chi,\alpha}$.

Case 4. $s \geq \frac{n}{2}$ and $s + l \geq n$ (Fig. 15d). Similar to Case 2, some αs on the inputs should be mapped to the outputs in a parallel way, and some αs on the inputs should be mapped to the outputs in a crossing way. Note that $s_0 = s - \frac{n}{2}$ and $s_1 = s + l - n$. Under the switch setting $W^{\frac{n}{2}}_{s_1,l_1,\frac{n}{2}-s_1-l_1;0,2,1}$, the subsegment x_0 (consisting of all αs) in the upper half of the inputs can be divided into two parts. The subsegment on the inputs with αs starting at $s_0 = s - \frac{n}{2}$ and ending at $\frac{n}{2} - 1$ is mapped to the outputs in a crossing way; and the subsegment on the inputs with αs starting at 0 and ending at $s_1 - 1$ is mapped to the outputs in a parallel way. In other words, $l = (\frac{n}{2} - s_0) + s_1 = n - s + s_1$ consecutive outputs have αs starting at $s = s_0 + \frac{n}{2}$ (in a circular way, modulo n), and the rest of outputs have χs, which is $C^n_{s,l;\chi,\alpha}$. □

APPENDIX C

DESCRIPTIONS OF THE DISTRIBUTED SELF-ROUTING ALGORITHMS

This appendix consists of Table 3, Table 4, Table 5, and Table 6, which provide formal descriptions of the distributed self-routing algorithms as discussed in Section 6.

ACKNOWLEDGMENTS

We would like to thank the anonymous referees of this article for their thoughtful and constructive comments. Yuanyuan Yang was supported by the U.S. Army Research Office under Grant No. DAAH04-96-1-0234 and by the U.S. National Science Foundation under Grant No. OSR-9350540 and MIP-9522532. A preliminary version of this work was presented at the 12th IEEE International Parallel Processing Symposium (IPPS/SPDP 1998).

REFERENCES

[1] D.K. Panda, "Issues in Designing Efficient and Practical Algorithms for Collective Communication on Wormhole-Routed Systems," *ICPP'95 Workshop Challenges for Parallel Processing,* pp. 8-15, 1995.

[2] L.M. Ni, "Should Scalable Parallel Computers Support Efficient Hardware Multicast?" *ICPP'95 Workshop Challenges for Parallel Processing,* pp. 2-7, 1995.

[3] J. Duato, S. Yalamanchili, and L.M. Ni, *Interconnection Networks: An Engineering Approach.* Los Alamitos, Calif.: IEEE CS Press, 1997.

[4] D. Nassimi and S. Sahni, "Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network," *J. ACM,* vol. 29, no. 3, pp. 642-667, July 1982.

[5] J. Turner, "Design of a Broadcast Packet Switching Network," *IEEE Trans. Comm.,* vol. 36, no. 6, pp. 734-743, 1988.

[6] T.T. Lee, "Nonblocking Copy Networks for Multicast Packet Switching," *IEEE J. Selected Areas in Comm.,* vol. 6, no. 9, pp. 1,455-1,467, 1988.

[7] C.-T. Lea, "A New Broadcast Switching Network," *IEEE Trans. Comm.,* vol. 36, no. 10, pp. 1,128-1,137, 1988.

[8] Y. Yang, G.M. Masson, "Nonblocking Broadcast Switching Networks," *IEEE Trans. Computers,* vol. 40, pp. 1,005-1,015, 1991.

[9] C. Lee and A.Y. Oruç, "Design of Efficient and Easily Routable Generalized Connectors," *IEEE Trans. Comm.,* vol. 43, nos. 2, 3, 4, pp. 646-650, 1995.

[10] Y. Yang and G.M. Masson, "Broadcast Ring Sandwich Networks," *IEEE Trans. Computers,* vol. 44, no. 10, pp. 1,169-1,180, Oct. 1995.

[11] D.M. Koppelman and A.Y. Oruç, "A Self-Routing Permutation Network," *J. Parallel and Distributed Computing,* vol. 10, no. 10, pp. 140-151, Oct. 1990.

[12] C.Y. Jan and A.Y. Oruç, "Fast Self-Routing Permutation Switching on an Asymptotically Minimum Cost Network," *IEEE Trans. Computers,* vol. 42, no. 12, pp. 1,469-1,479, Dec. 1993.

[13] S. Lee and M. Lu, "New Self-Routing Permutation Networks," *IEEE Trans. Computers,* Vol. 43, no. 11, pp. 1,319-1,323, Nov. 1994.

[14] W.-J. Cheng and W.-T. Chen, "A New Self-Routing Permutation Network," *IEEE Trans. Computers,* vol. 45, no. 5, pp. 630-636, May 1996.

[15] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, 1993.

Yuanyuan Yang received the BEng and MS degrees in computer engineering from Tsinghua University, Beijing, China, in 1982 and 1984, respectively, and the MSE and PhD degrees in computer science from The Johns Hopkins University, Baltimore, Maryland in 1989 and 1992, respectively. She is currently an associate professor of computer engineering at the State University of New York at Stony Brook. Dr. Yang was a faculty member in the Department of Computer Science, the University of Vermont at Burlington from 1992-1999 (as an associate professor from 1998-1999). Dr. Yang's research interests include parallel and distributed computing and systems, high speed networks, optical networks, high performance computer architecture, and fault-tolerant computing. She has published extensively in archival journals and refereed conference proceedings in these areas. Dr. Yang holds two U.S. patents in the area of multicast communication networks, with four more patents pending. Her research has been supported by the U.S. Army Research Office and the National Science Foundation. She has served on the program/organizing committees of a number of international conferences. Dr. Yang is a senior member of the IEEE and a member of the ACM, the IEEE Computer Society, and the IEEE Communication Society.

Jianchao Wang received the BEng degree in computer engineering from Tsinghua University, Beijing, China, in 1982, and the MS and PhD degrees in computer science from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1985 and 1988, respectively. He is currently a principal member of technical staff at GTE Laboratories Incorporated in Waltham, Massachusetts. Before he joined GTE Labs, Dr. Wang worked at the Institute of Computing Technology, Chinese Academy of Sciences, The Johns Hopkins University in Baltimore, and Legent Corporation in Marlboro, Massachusetts. Dr. Wang has received a number of excellence/achievement awards from GTE and Legent, and has five U.S. patents granted or pending. Dr. Wang's research interests include IP telephony, databases, programming languages, computer communication networks, computer algorithms, and fault-tolerant computing. Dr. Wang is a member of the IEEE Computer Society and the ACM.