CHAPTER 8

BANYAN-BASED SWITCHES

The very early theoretical work on multistage interconnection networks (MIN) was done in the context of circuit-switched telephone networks [1, 2]. The aim was to design a nonblocking multistage switch with a number of crosspoints less than a single-stage crossbar switch. After many such networks were studied and introduced for interconnecting multiple processors and memories in parallel computer systems, several types of them such as banyan and shuffle-exchange networks were proposed [3–5] as a switching fabric because several cells could be routed in parallel and the switching function could be implemented regularly in hardware.

In this chapter, we describe the banyan-family of switches, which have attracted many researchers for more than two decades now to build interconnection networks. Section 8.1 classifies banyan-family switch architectures into different categories based on their nature and property. Section 8.2 describes Batcher-sorting network switch architecture. Section 8.3 introduces output-contention resolution algorithms in banyan-family switches. Section 8.4 describes the Sunshine switch which extends the Batcher-banyan switching architecture. Section 8.5 describes some work on deflection routing over banyan-family networks. Section 8.6 introduces a self-route copy network where the nonblocking property of the banyan network is generalized to support multicasting.

8.1 BANYAN NETWORKS

The banyan class of interconnection networks was originally defined by Goke and Lipovski [6]. It has the property that there is exactly one path from any input to any output. Figure 8.1 shows four networks belonging to this class: the shuffle-exchange network (also called the omega network), the reverse shuffle-exchange network, the banyan network, and the baseline network.

High Performance Switches and Routers, by H. Jonathan Chao and Bin Liu
Copyright © 2007 John Wiley & Sons, Inc.

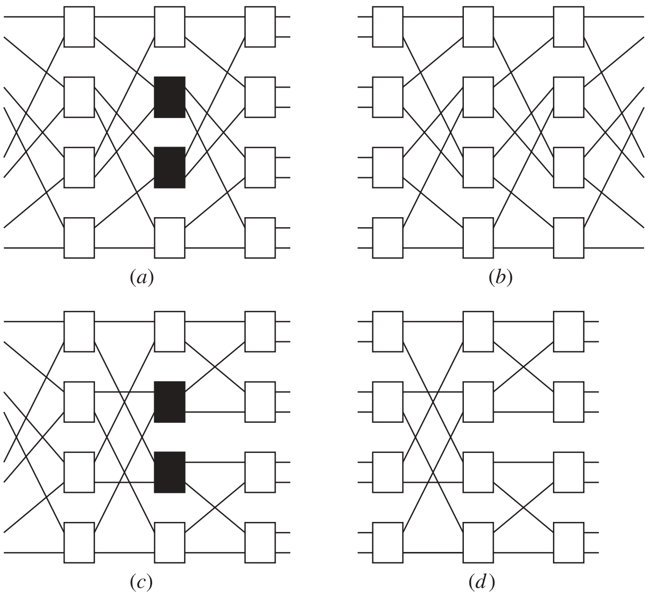


Figure 8.1 Four different banyan networks: (*a*) shuffle-exchange (omega) network; (*b*) reverse shuffle-exchange network; (*c*) banyan network; (*d*) baseline network. We can see that (*a*) and (*c*) are isomorphic by interchanging two shaded nodes in the figures.

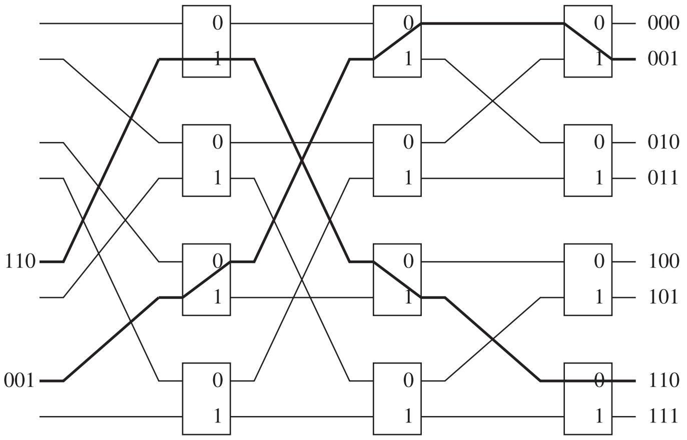


Figure 8.2 8×8 banyan network.

The common principal properties of these networks are: (1) they consist of $n = \log_2 N$ stages and $N/2$ nodes per stage;[1] (2) they have the self-routing property such that the unique n-bit destination address can be used to route a cell from any input to any output, each bit for a stage; and (3) their regularity and interconnection pattern are very attractive for VLSI implementation. Figure 8.2 shows a routing example in an 8×8 banyan network where the bold lines indicate the routing paths. On the right-hand side, the address of each output destination is labeled as a string of n bits, $b_1 \cdots b_n$. A cell's destination address is encoded

[1]A regular $N \times N$ network can also be constructed from identical $b \times b$ switching nodes in k stages where $N = b^k$.

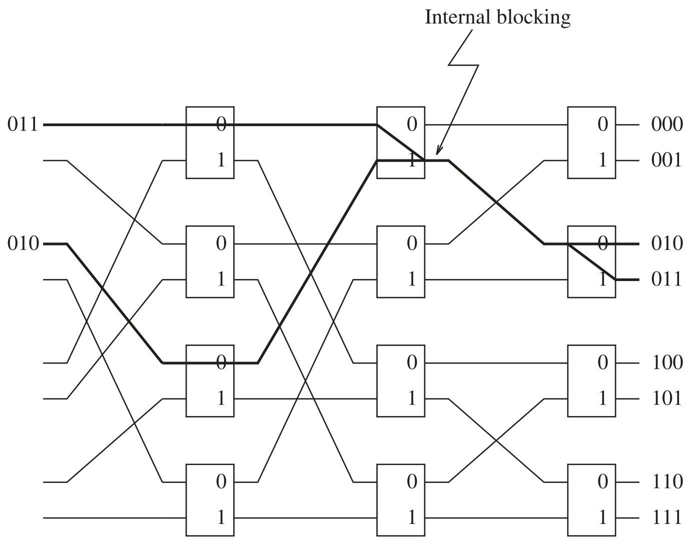


Figure 8.3 Internal blocking in an 8 × 8 banyan network.

into the header of the cell. In the first stage, the most significant bit b_1 is examined. If it is a 0, the cell will be forwarded to the upper outgoing link; if it is a 1, the cell will be forwarded to the lower outgoing link. In the next stage, the next most significant bit b_2 will be examined and the routing is performed in the same manner.

The internal blocking refers to the case where a cell is lost due to the contention on a link inside the network. Figure 8.3 shows an example of internal blocking in an 8 × 8 banyan network. However, the banyan network will be internally nonblocking if both conditions below are satisfied:

- There is no idle input between any two active inputs;
- Output addresses of the cells are in either an ascending order or a descending order.

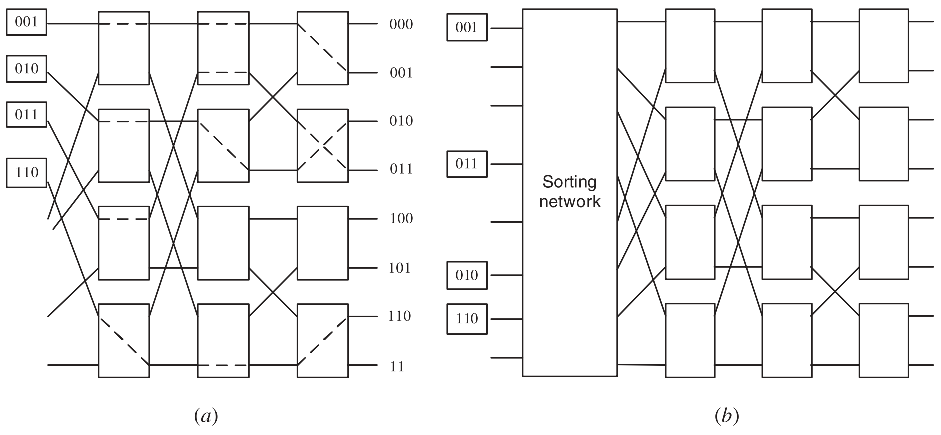
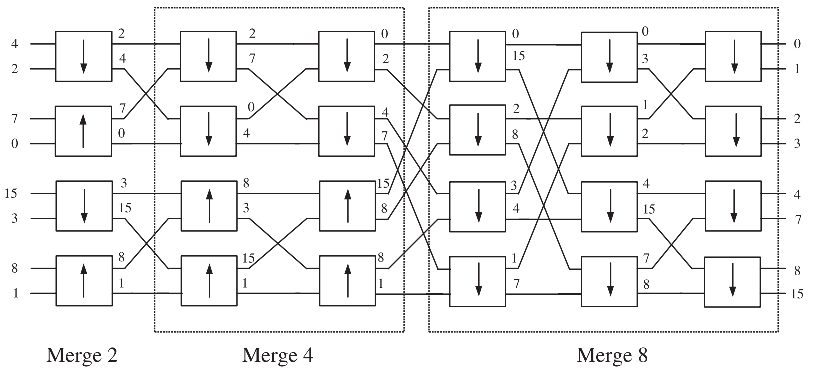


Figure 8.4 (*a*) Example showing the banyan network is nonblocking for sorted inputs. (*b*) Nonblocking sort-banyan network.

See Figure 8.4. Consider the banyan network to be preceded by a network that concentrates the cells and sorts the cells according to their output destinations. The overall sort-banyan network will be internally nonblocking.

8.2 BATCHER-SORTING NETWORK

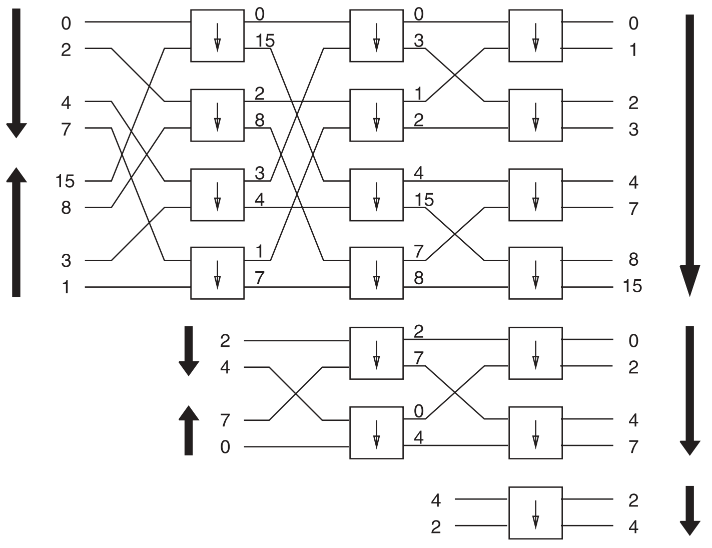
A sorting network is formed by a series of merge networks of different sizes. Figure 8.5*a* shows an 8 × 8 Batcher-sorting network [7] consisting of merge networks of three different sizes. A merge network (see Figure 8.5*b*) is built by 2 × 2 sorting elements in stages, and the pattern of exchange connections between each pair of adjacent stages is the same as in



- Batcher sorting network consists of multiple merge networks.

-Total numbers of stages= $(1 + 2 + \cdots + \log_2 N) = (\log_2 N)(1 + \log_2 N)/2$

(*a*)



(*b*)

Figure 8.5 Basic structure of a Batcher-sorting network. (*a*) Batcher-sorting network; (*b*) Corresponding merge networks.

a banyan network. We can observe that, if the order of the destinations of the first half input cells is ascending and that of the second half is descending, then the merge network will sort the cells into an ascending list at the outputs. An 8×8 sorting network will be formed if an 8×8 merge network is preceded by two 4×4 merge networks and four 2×2 merge (sorting) elements. A completely random list of eight input cells will be first sorted into four sorted lists of two cells, then two sorted lists of four cells, and finally a sorted list of eight cells.

A $N \times N$ merge network consists of $\log_2 N$ stages and $N \log_2 N/2$ elements. A sorting network has $1 + 2 + \cdots + \log_2 N = \log_2 N(\log_2 N + 1)/2$ stages and $N \log_2 N\ (\log_2 N + 1)/4$ elements. Figure 8.6 shows a 64×64 Batcher-banyan switch network, where the last six stages of switch elements belong to the banyan network.

8.3 OUTPUT CONTENTION RESOLUTION ALGORITHMS

8.3.1 Three-Phase Implementation

The following three-phase algorithm is a solution for output contention resolution in a Batcher-banyan switch (see Figure 8.7).

In the first (arbitration) phase of the algorithm, each input port i sends a short request only, consisting of a source-destination pair, to the sorting network where the requests are sorted in nondecreasing order according to the destination address. In other words, conflicting requests are sorted at adjacent positions, and a request wins the contention only if its destination is different from the one above it in the sorted list.

As the input ports do not know the result of the arbitration, the requests that won the arbitration must send an acknowledgment to the input ports via an interconnection network in the second phase (the so-called acknowledgment phase). The feedback network in Figure 8.7*b* consists of N fixed connections, each from an output of the Batcher network to the corresponding input of the Batcher network. Each acknowledgment carries the source that has won the contention back to an input of the Batcher network. These acknowledgments (sources) are routed through the entire Batcher-banyan network at distinct outputs according to the source address. When these acknowledgments are feedbacked to the inputs through an identical fixed network, each input port knows if it has won the contention. The input ports that finally receive an acknowledgment are guaranteed conflict-free output.

These input ports then transmit the full cell in the third and final phase (see Figure 8.7*c*) through the same Batcher-banyan network. Input ports that fail to receive an acknowledgment retain the cell in a buffer for the retry in the next time slot when the three-phase cycle is repeated.

8.3.2 Ring Reservation

A Batcher-banyan cell switch design with ring reservation is shown in Figure 8.8 [8]. The switch comprises the Batcher-banyan switch fabric, several switch interfaces, and a ring head-end (RHE) and timing generator.

A switch interface supports ring reservation and provides input cells buffering, synchronization of cells sent to the switch fabric, and output cells buffering. Cells entering the switch are buffered in a FIFO until they can participate in the reservation procedure. When an output is successfully reserved, the cell is delivered to the switch fabric at the beginning of the next cell cycle, and the next queued cell can begin to participate in the

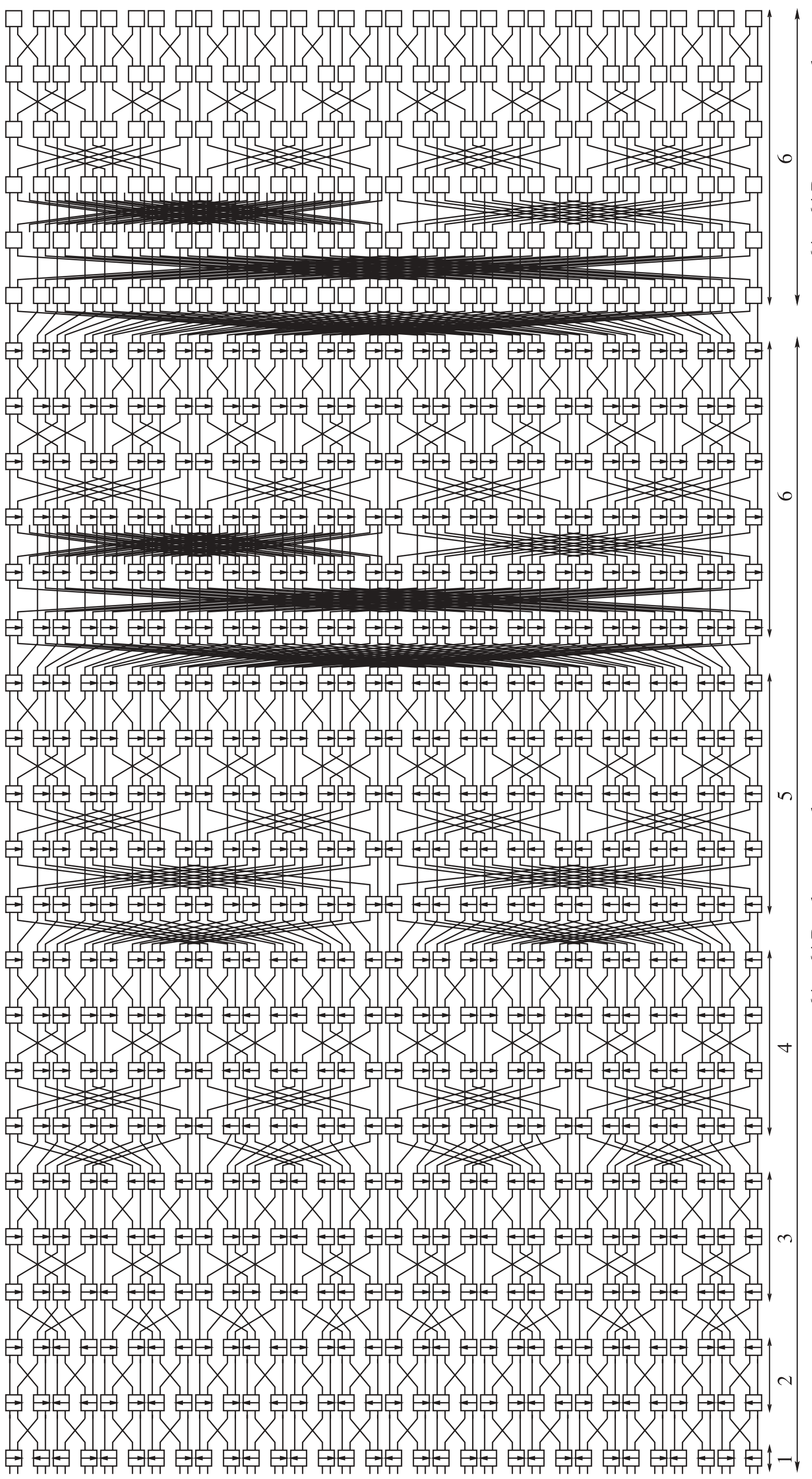
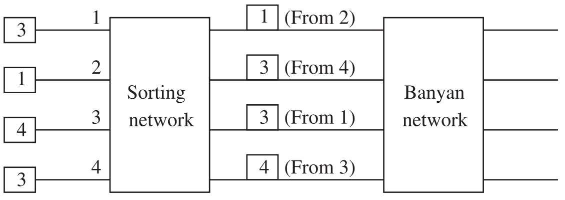
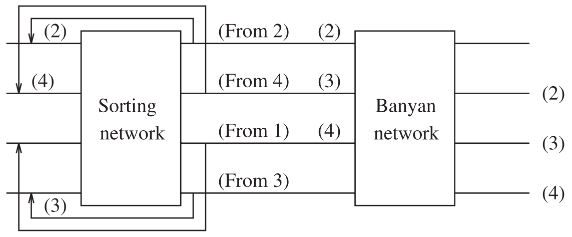


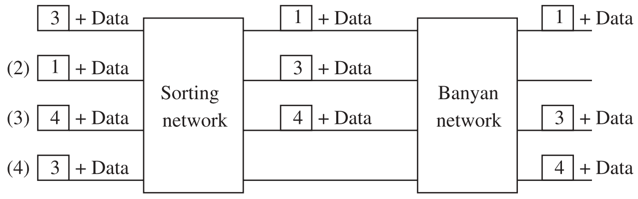
Figure 8.6 64 × 64 Batcher-banyan switch network.



(*a*) - Send source-destination pair through sorting network
- Sort destination in non-decreasing order
- Purge adjacent requests with same destination



(*b*) - Send ACK with source to ports winning contention
- Route ACK through Batcher-banyan network



(*c*) - Acknowledged ports send cells through Batcher-banyan network
- Cells not acknowledged are buffered and retry in the next slot

Figure 8.7 Three-phase algorithm. (*a*) Phase I: send and resolve request; (*b*) Phase II: acknowledge winning ports; (*c*) Phase III: send with data.

reservations. When the cell emerges from the output of the switch fabric, it is buffered in the interface before being transmitted to its destination.

The RHE provides two switch synchronization signals: bit clock and cell cycle start; and three ring reservation signals: ring clock, ring data, and the ring sync. The ring data is a series of output reservation bits, and the ring sync signal indicates the position of the first output port in the ring data series. These two signals are circulated through the RHE and switch interfaces, one bit at a time, during the reservation process. Ring reservation is performed at the beginning of each cell cycle after every ring interface has the header of the oldest cell copies. The ring data in the RHE and each ring interface is cleared (idle) at every cell cycle start. The ring data series then begins to circulate through the interface bit-by-bit. Each interface maintains a port counter that is incremented in every ring data bit time. The port counter is compared to the destination of the oldest cell in every bit time indicating if the cell is destined for the output in the next bit time. During each ring data bit time, each switch interface examines both the ring sync and the ring data bit. If the ring sync signal is

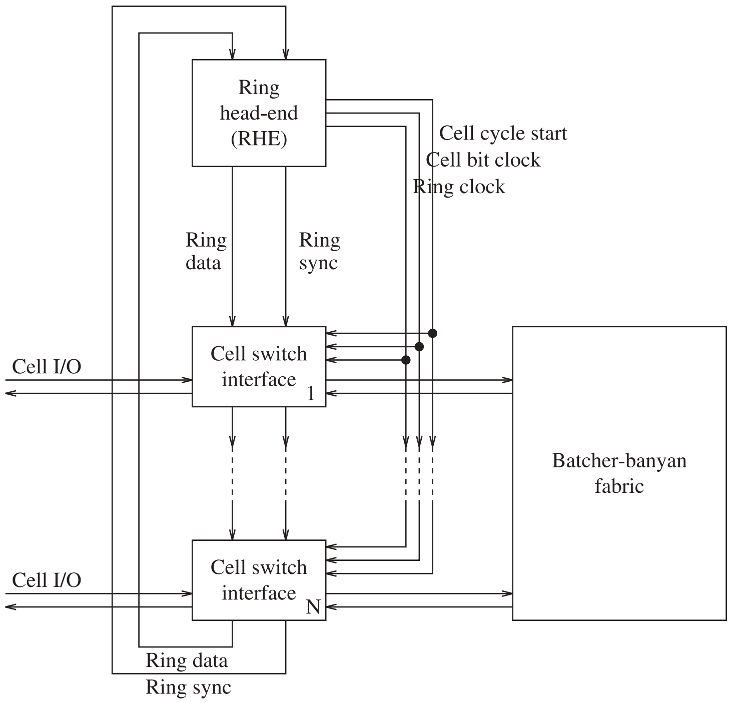


Figure 8.8 Batcher-banyan switch with ring reservation.

true, which means the next ring data bit corresponds to the first output, then the port counter is reset in the next bit time. If the destination of the cell matches with the port counter and the ring data bit is idle, the switch interface writes ‘busy’ on the ring to show that the output has been occupied in the next switch cycle. If the ring data bit is already BUSY, or if the port counter does not match the destination of the oldest cell, the ring data bit is left unchanged. Since each interface makes no more than one reservation per switch cycle, no collisions can take place in the switch fabric. While the ring reservation is being performed, the cells reserved in the previous switch cycle are transmitted to the switch fabric.

As shown in Fig. 8.9, at the first time slot, the output port addresses of cells from input ports 1 and 5 are matched, some checks are used to indicate that the cells can be sent to

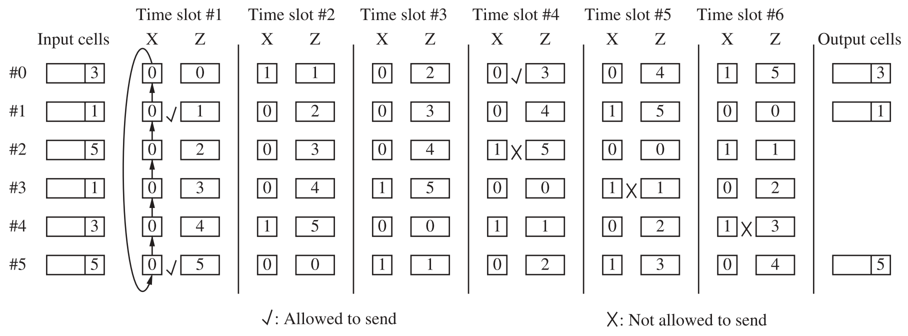


Figure 8.9 Implementation of the ring reservation scheme.

these output ports. The token bits x_1 and x_5 are set to one to indicate that output ports 1 and 5 are already reserved. All the token bits are shifted up one bit and the counter values are also modulo-increased by one for the second time slot. There are no matches found at the second and the third time slots. At the fourth time slot, the output port addresses of cells from input ports 0 and 2 are matched. Since output port 5 is already reserved for the cell in the previous time slot, which is indicated by the value of the token bit x_2, the cell at input port 2 cannot be sent. Similarly, for the fifth and the sixth time slots, cells at input ports 3 and 4 cannot be sent to output ports 1 and 3, respectively, since those output ports are already reserved in the previous time slots. At the end, cells from the input ports that are checked are the ones winning the contention.

In this example, since there are six input ports, the arbitration cycle can be completed within six time slots. This scheme uses the serial mechanism and, in general, the arbitration cycle can be done within the N-bit time slot, where N is the number of input/output ports of the switch. This will become the bottleneck when the number of ports of the switch is large. However, by arbitrarily setting the appropriate values for the counters prior to the arbitration, this scheme provides fairness among the input ports. Another advantage of this scheme is that it can be employed at the input of any type of switch fabric.

8.4 THE SUNSHINE SWITCH

Sunshine switch [9] uses the combination of a Batcher-sorting network and parallel banyan routing networks to provide more than one path to each destination. Figure 8.10 shows a block diagram of the architecture. The k parallel banyan routing networks provide k independent paths to each output. If more than k cells request a particular output during a time slot, then some excess cells are overflowed into a shared recirculating queue and then resubmitted to the switch at dedicated input ports. The recirculating queue consists of T parallel loops and T dedicated inputs to the Batcher-sorting network. Each recirculating loop can hold one cell. A delay block is put within the loops to align recirculated cells with those new arrivals from the input port controllers (IPCs) in the next time slot. During each time slot, the Batcher network sorts newly arrived and recirculated cells in the order of destination address and priority. This enables the trap network to resolve output port contention by selecting the k highest priority cells for each destination address. Since there are k parallel banyan networks, each output can accept k cells in a time slot. When there are more than k cells destined for an output, the excess will enter the recirculating queue. The concentrator and the selector will direct the excess cells to the recirculating loops, while those cells selected for routing will be forwarded to the banyan networks.

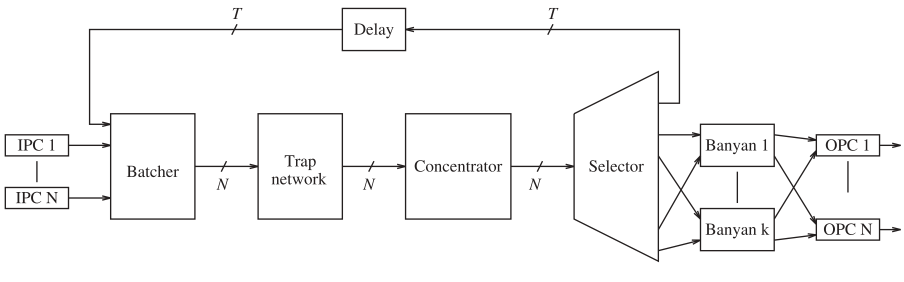


Figure 8.10 Block diagram of the Sunshine switch.

Each cell is inserted with a control header at the input port controller. The header format is shown in Figure 8.11. It contains two control fields: a routing field and a priority field, both are ordered starting with the most significant bit. In the routing field, the first bit is a cell activity bit to indicate if the cell contains valid information ($A = 1$), or the cell is empty ($A = 0$). Then the destination address (DA) field identifying the desired output port follows. The priority field consists of the quality of service (QoS) indicator and the internal switch priority (SP). The QoS field distinguishes cells from higher priority services such as circuit emulation, from lower priority services such as connectionless service, and ensures higher priority cells to be routed before lower priority cells when conflicts arise. The SP field is used internally by the switch to indicate the number of time slots that a cell has been delayed, and gives higher priority to recirculate cells. This guarantees that cells from a given source will be routed in sequence.

When the cells are sorted, they are arranged in ascending order of destination address. The priority field, where a higher numerical value represents a higher priority level, appears as an extension of the routing field. This causes cells destined for the same output to be arranged in the descending order of priority. In the trap network, the address of every cell is compared with that of the cell k positions above. If a cell has the same address as the cell k positions above, which indicates that there are at least k cells with higher priority, then the cell is marked to be recirculated, and its routing field is interchanged with the priority field because the priority field is more concerned with the following concentration sorting network than the recirculation loss. Otherwise, the cell is one of the k (or less) highest priority cells for the address, and is set to be routed.

In the Batcher concentration network, there are two groups of cells, one group of cells to be routed and the other to be recirculated, each is sorted into a contiguous list, respectively. Then the group of cells to be routed forms a list in ascending order of the destination address, to avoid subsequent internal blocking in the banyan networks. The group of cells to be recirculated is sorted into a separate contiguous list according to the priority. If the recirculating queue overflows, the cells with lower priorities are more likely to be dropped than those with higher priorities.

Cells are then directed to the selector that differentiates two groups of cells towards k banyan networks and to T recirculators, respectively. Cells that enter the recirculators will have their routing and priority fields interchanged back into the original format. Their priority, SP, is incremented as the cell has been recirculated.

The outputs of the selectors are spread among the k banyan networks by connecting every kth output to the same banyan network. This ensures every two cells destined for the same output are separated into different banyan networks. The cells in each banyan network still constitute a contiguous list destined for distinct outputs, which satisfies the

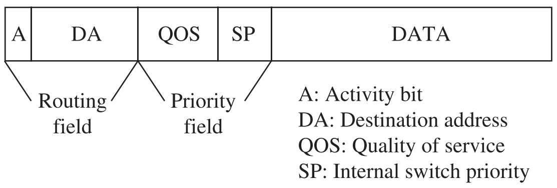


Figure 8.11 Header format.

nonblocking condition in a banyan network. Every cell then reaches the desired output of a banyan network, and all corresponding outputs are grouped together to an output queue in the output port controller (OPC).

8.5 DEFLECTION ROUTING

8.5.1 Tandem Banyan Switch

Figure 8.12 shows the tandem banyan switching fabric (TBSF) [10]. It consists of multiple banyan networks in series. While two cells contend at any node in a banyan network, one of them will just be deflected to the wrong output of the node and finally arrive at an incorrect destination of the banyan network. The deflected cell is then forwarded to the next banyan network. This process continues again and again until the cell reaches the desired output or it gets out of the last banyan network at an undesired output and is regarded as lost. Each output of every banyan network is connected to the corresponding output buffer. A cell is marked misrouted when it gets deflected in a banyan network to distinguish from those properly routed cells and to avoid affecting their routing at later stages within the network. At outputs of each banyan network, the cells that have reached their respective destinations are extracted from the fabric and placed in output port buffers. Note that the load of successive banyan networks decreases and so does the likelihood of conflicts. With a sufficiently large number of banyan networks, it is possible to reduce the cell loss to desired levels. Numerical results show that each additional banyan network improves the cell loss probability by one order of magnitude.

The operation of the TBSF is described as follows. A switching header is appended to each cell when it enters the switching fabric, and it comprises the following four fields:

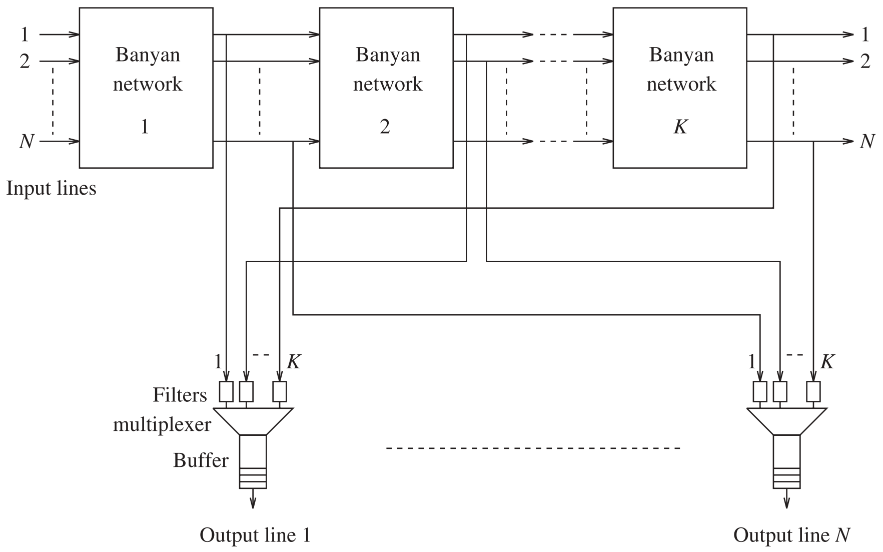


Figure 8.12 Tandem banyan switching fabric.

- *Activity Bit a*: It indicates whether the slot contains a cell ($a = 1$), or is empty ($a = 0$).
- *Conflict Bit c*: It indicates whether the cell has already been misrouted at some previous stage of the present network ($c = 1$), or not ($c = 0$).
- *Priority Field P*: It is optional and is used if multiple priority is supported over the switch.
- *Address Field D*: It contains the destination address, $d_1, d_2, \ldots, d_n$ ($n = \log_2 N$).

The state of a switching element at stage s of a banyan network is primarily determined by three bits in each header of the two input cells; namely, a, c, and d_s. If multiple priority is supported, then the switch state also depends on the priority field P. The algorithm is as follows, where the bits are indexed by 1 and 2 corresponding to the two input cells.

1. If $a_1 = a_2 = 0$, then no action is taken, that is, leaves the switch in the present state;
2. If $a_1 = 1$ and $a_2 = 0$, then set the switch according to d_{s1};
3. If $a_1 = 0$ and $a_2 = 1$, then set the switch according to d_{s2};
4. If $a_1 = a_2 = 1$, then
   (a) If $c_1 = c_2 = 1$, then no action is taken;
   (b) If $c_1 = 0$ and $c_2 = 1$, then set the switch according to d_{s1};
   (c) If $c_1 = 1$ and $c_2 = 0$, then set the switch according to d_{s2};
   (d) If $c_1 = c_2 = 0$, then
      i. If $P_1 > P_2$, then set the switch according to d_{s1};
      ii. If $P_1 < P_2$, then set the switch according to d_{s2};
      iii. If $P_1 = P_2$, then set the switch according to either d_{s1} or d_{s2}.
      iv. If one of the cells has been misrouted, then set its conflict bit to 1.

In order to minimize the number of bits to be buffered at each stage to perform the above algorithm, thus to minimize the latency incurred at each stage, the address bit is placed in the first position of the address field. This can be done by cyclically shifting the address field by one bit at each stage. It is then possible to keep the latency at each stage, as low as 3 bit times without considering multiple-priority support, and it is constant over all stages.

It is simple to differentiate successful cells and deflected cells at outputs of each banyan network with the conflict bit: if $c = 0$, then the cell has been properly routed; if $c = 1$, then the cell has been misrouted. A cell with $c = 0$ is accepted by the output buffer, and is rejected by the next banyan network by setting the activity bit in that slot to 0. Conversely, a cell with $c = 1$ is ignored by the output buffer, but is accepted by the following banyan network with its conflict bit reset to 0 for further routing.

All cells entering the tandem banyan switch fabric in the same time slot must be bit-synchronized throughout the entire fabric. If ignoring the propagation delay, the delay for each cell in a banyan network is constant and is equal to n times the processing delay at a switching element, or the time difference between two cells emerging from adjacent banyan networks. In order to have all cells from different banyan networks arriving at an output buffer at the same time, an appropriate delay element can be placed between each output and each banyan network.

In addition, the output buffer memory should have an output bandwidth equal to V bit/s and an input bandwidth equal to KV bit/s to accommodate as many as K cells arriving in the same slot.

8.5.2 Shuffle-Exchange Network with Deflection Routing

Liew and Lee [11] considered an $N \times N$ shuffle-exchange network (SN) with $n = \log_2 N$ stages, each consisting of $(N/2)2 \times 2$ switch elements. Figure 8.13 shows an 8×8 SN. Switch nodes in each stage are labeled by an $(n-1)$-bit binary number from top to bottom. The upper input (output) of a node is labeled with a 0, and the lower input (output) is labeled with a 1. A cell will be forwarded to output 0(1) at stage i if the ith most significant bit of its destination address is 0(1). The interconnection between two consecutive stages is called 'shuffle exchange'. The output a_n of node $X = (a_1 a_2 \cdots a_{n-1})$ is connected to the input a_1 of node $Y = (a_2 a_3 \cdots a_n)$ of the subsequent stage. The link between node X and node Y is labeled as $\langle a_n, a_1 \rangle$. The path of a cell from input to output is completely determined by the source address $S = s_1 \cdots s_n$ and the destination address $D = d_1 \cdots d_n$. It can be expressed symbolically as follows:

$$\begin{aligned}
&S = s_1 \cdots s_n \\
&\xRightarrow{\langle -, s_1 \rangle} \quad (s_2 \cdots s_n) \xRightarrow{\langle d_1, s_2 \rangle} (s_3 \cdots s_n d_1) \\
&\xRightarrow{\langle d_2, s_3 \rangle} \quad \cdots \xRightarrow{\langle d_{i-1}, s_i \rangle} (s_{i+1} \cdots s_n d_1 \cdots d_{i-1}) \\
&\xRightarrow{\langle d_i, s_{i+1} \rangle} \quad \cdots \xRightarrow{\langle d_{n-1}, s_n \rangle} (d_1 \cdots d_{n-1}) \\
&\xrightarrow{\langle d_n, 0 \rangle} \quad d_1 \cdots d_n = D.
\end{aligned}$$

The node sequence along the path is embedded in the binary string $s_2 \cdots s_n d_1 \cdots d_{n-1}$, represented by an $(n-1)$-bit window moving one bit per stage from left to right.

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

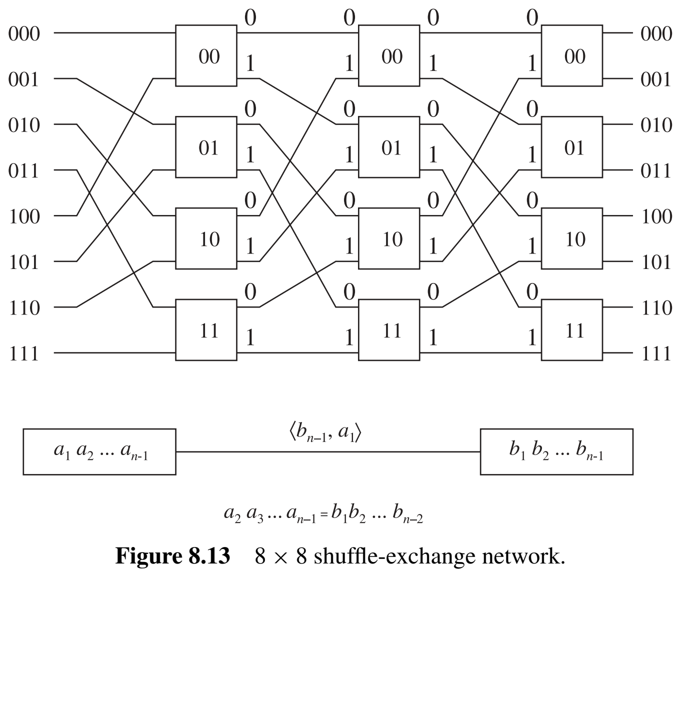

000
001
010
011
100
101
110
111
00
01
10
11
0
1
000
001
010
011
100
101
110
111
$a_1\ a_2 \ldots a_{n-1}$
$\langle b_{n-1}, a_1 \rangle$
$b_1\ b_2 \ldots b_{n-1}$
$a_2\ a_3 \ldots a_{n-1} = b_1 b_2 \ldots b_{n-2}$

Figure 8.13 8×8 shuffle-exchange network.

algorithm as follows:

$$\underset{\textit{input label } x_n}{(r_1 \cdots r_k, x_1x_2 \cdots x_{n-1})} \xrightarrow{\textit{exchange}} \underset{\textit{output label } r_k}{(r_1 \cdots r_{k-1},\, x_1x_2 \cdots x_{n-1})}$$

$$\xrightarrow[\langle r_k, x_1\rangle]{\textit{shuffle}} (r_1 \cdots r_{k-1}, x_2 \cdots x_{n-1}r_k).$$

Notice that the routing bit used in the switching node is removed from the routing tag after each stage, before the node label is shuffled to the next stage. Finally, the cell will reach the state $(d_nd_1 \cdots d_{n-1})$, from which the following 2×2 element will switch the cell to the destination.

When a contention occurs at a switch node, one of cells will be successfully routed while the other one will be deflected to the wrong output. As a result, only the non-deflected cells can reach their desired outputs eventually. The deflected cells can restart routing (with routing tag reset to $d_n \cdots d_1$) again at the deflection point, and if the SN is extended to consist of more than n stages, those deflected cells could reach the destination at later stages. As some cells will reach their destinations after fewer numbers of stages than others, a multiplexer is needed to collect cells that reach physical links of the same logical address at different stages. A cell will eventually reach its destination address with good probability provided that the number of stages L is sufficiently large. If it cannot reach the destination after L stages, it is considered lost.

8.5.3 Dual Shuffle-Exchange Network with Error-Correcting Routing

The error-correcting SN is highly inefficient, especially when n is large. This is because routing of the cell must be restarted from the beginning whenever it is deflected. This is illustrated by the state-transition diagram in Figure 8.14*a*, where the state is the distance or the number of stages away from destination. A desired network should be one with the state-transition diagram shown in Figure 8.14*b*, in which the penalty is only one step backward.

An example is the dual shuffle-exchange network that consists of a shuffle exchange network and an unshuffle-exchange network (USN) [11]. An 8×8 USN is shown in Figure 8.15.

It is the mirror image of the SN. Routing in successive stages is based on the least significant bit through the most significant bit. Using a numbering scheme similar to that in

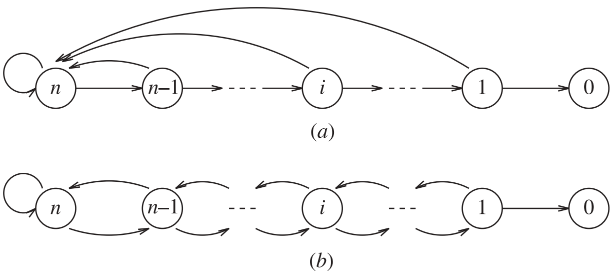


Figure 8.14 (*a*) State-transition diagram of a cell in the shuffle-exchange network, where the distance from destination is the state; (*b*) One-step penalty state transition diagram.

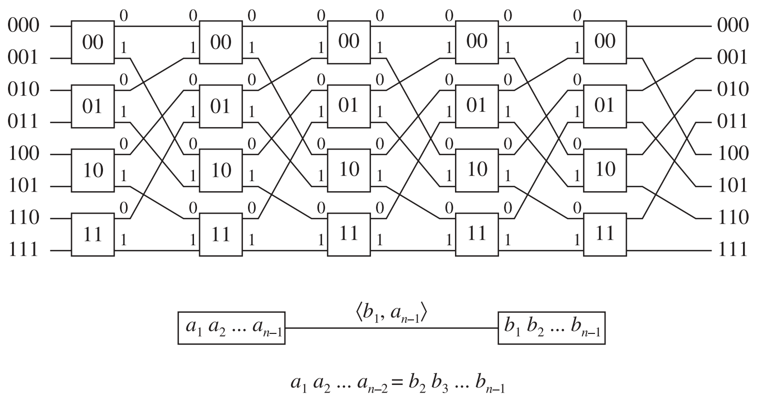

000
001
010
011
100
101
110
111
00
01
10
11
$a_1\, a_2 \ldots a_{n-1}$
$\langle b_1, a_{n-1}\rangle$
$b_1\, b_2 \ldots b_{n-1}$
$a_1\, a_2 \ldots a_{n-2} = b_2\, b_3 \ldots b_{n-1}$

Figure 8.15 8 × 8 unshuffle-exchange network with five stages.

SN, the path of a cell with source address $S = s_1 \cdots s_n$ and destination address $D = d_1 \cdots d_n$ can be expressed by

$$
\begin{aligned}
&S = s_1 \cdots s_n \\
&\xRightarrow{\langle -, s_n\rangle} \quad (s_1 \cdots s_{n-1}) \quad \xRightarrow{\langle d_n, s_{n-1}\rangle} \quad (d_n s_1 \cdots s_{n-2}) \\
&\xRightarrow{\langle d_{n-1}, s_{n-2}\rangle} \quad \cdots \quad \xRightarrow{\langle d_{i+2}, s_{i+1}\rangle} \quad (d_{i+2} \cdots d_n s_1 \cdots s_i) \\
&\xRightarrow{\langle d_{i+1}, s_i\rangle} \quad \cdots \quad \xRightarrow{\langle d_2, s_1\rangle} \quad (d_2 \cdots d_n) \\
&\xrightarrow{\langle d_1, 0\rangle} \quad d_1 \cdots d_n \quad = D.
\end{aligned}
$$

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

$$
\begin{aligned}
\underset{\textit{input label } x_n}{(r_1 \cdots r_k, x_1 x_2 \cdots x_{n-1})} \quad &\xRightarrow{\textit{exchange}} \quad \underset{\textit{output label } r_k}{(r_1 \cdots r_{k-1}, x_1 x_2 \cdots x_{n-1})} \\
&\xRightarrow[\langle r_k, x_{n-1}\rangle]{\textit{unshuffle}} \quad (r_1 \cdots r_{k-1}, r_k x_1 \cdots x_{n-2})
\end{aligned}
$$

At the last stage, the cell is in state $(-d_1 d_2 \cdots d_n)$, reaches its destination.

Suppose a USN is overlaid on top of a SN, and each node in the USN is combined with its corresponding node in SN such that a cell at any of the four inputs of the node can access any of the outputs of the node. The shuffle and the unshuffle interconnections between adjacent stages (nodes) compensate each other, so that the error caused by deflection in the SN can be corrected in the USN in only one step. See Figure 8.16, where cell A enters a SN from input 010 to output 101, and cell B, from input 100 to output 100. They collide at the second stage when they both arrive at node 01 and request for output 0. Suppose cell B wins the contention and cell A is deflected to node 11 in the third stage. Imagine cell A is

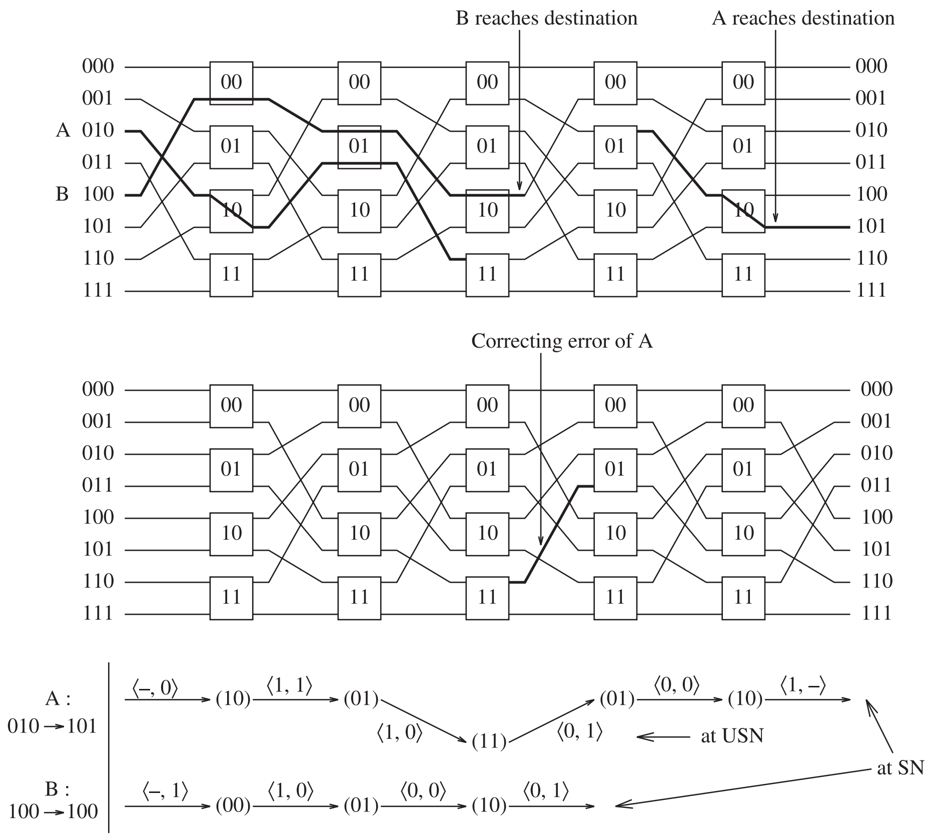
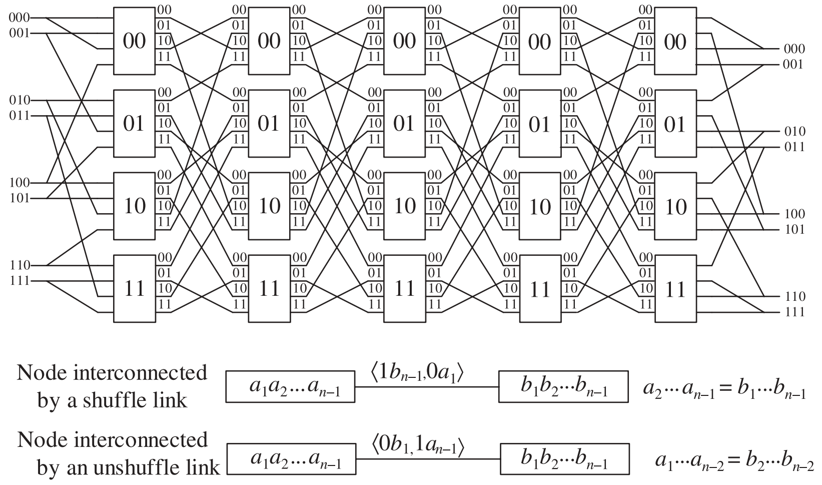


Figure 8.16 Deflection error in SN is corrected with USN.

moved to the companion node 11 in the corresponding USN, and is switched to output 0. Then it returns at reach node 01, the same node (label) when error occurred, in two stages. At this point, the deflection error has been corrected and cell *A* can continue on its normal path in the SN. Intuitively, any incorrect routing operation is undone in the SN by a reverse routing operation in the USN.

The above procedure can be formulated more rigorously as follows. Consider a cell in state $(r_1 \cdots r_k, x_1 \cdots x_{n-1})$, the cell should be sent out on link $\langle r_k, x_1 \rangle$ in the SN. Suppose it is deflected to link $\langle \bar{r}_k, x_1 \rangle$ instead and reaches node $(x_2 \cdots x_{n-1}\bar{r}_k)$ in the next stage. The error correction starts by attaching the bit x_1 to the routing tag instead of removing the bit r_k, so that the state of the cell will be $(r_1 \cdots r_k x_1, x_2 \cdots x_{n-1}\bar{r}_k)$ in the next stage. Then the cell is moved to the companion node in the USN to correct the error. If the cell is successively routed this time, it will be sent out on link $\langle x_1, \bar{r}_k \rangle$ and return to the previous state $(r_1 \cdots r_k, x_1 \cdots x_{n-1})$. Similarly, an error occurring in the USN can also be fixed in one step with the SN. In general, a cell in the SN may also be deflected to a link in the USN and vice versa, and consecutive deflections can occur. A simple algorithm to take these considerations into account is described in the following.



000
001
010
011
100
101
110
111
00
01
10
11
Node interconnected by a shuffle link
$a_1a_2...a_{n-1}$
$\langle 1b_{n-1}, 0a_1\rangle$
$b_1b_2...b_{n-1}$
$a_2...a_{n-1} = b_1...b_{n-1}$
Node interconnected by an unshuffle link
$a_1a_2...a_{n-1}$
$\langle 0b_1, 1a_{n-1}\rangle$
$b_1b_2...b_{n-1}$
$a_1...a_{n-2} = b_2...b_{n-2}$

Figure 8.17 8 × 8 dual-shuffle exchange network.

First of all, the companion 2 × 2 switch elements in the SN and in the USN are merged to form 4 × 4 switch elements to allow cells to be switched between the SN and the USN. Figure 8.17 shows a dual shuffle-exchange network built with 4 × 4 switch elements. A new labeling scheme is used. The four inputs (outputs) of a switch node are labeled by 00, 01, 10, 11 from top to bottom. Outputs 00 and 01 are connected to the next stage according to an unshuffling pattern, while outputs 10 and 11 are connected to the next stage according to a shuffling pattern. On the other hand, inputs 00 and 01 are connected to the previous stage according to a shuffling pattern, while inputs 10 and 11 are connected to the previous stage according to an unshuffling pattern. A link with label $\langle 1a, 0b\rangle$ is an unshuffle link and a link with label $\langle 0a, 1b\rangle$ 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 $\langle 0b_1, 1a_{n-1}\rangle$ if $a_1 \cdots a_{n-2} = b_2 \cdots b_{n-1}$, and by a shuffle link $\langle 1b_{n-1}, 0a_1\rangle$ if $a_2 \cdots a_{n-1} = b_1 \cdots b_{n-2}$.

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

The state of a cell at any particular time is denoted by $(c_1r_1 \cdots c_kr_k, x_1 \cdots x_{n-1})$. There are two possible regular transitions at a switch node; the cell 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

$$(c_1r_1 \cdots c_kr_k, x_1 \cdots x_{n-1}) \longrightarrow \left\{ \begin{array}{ll} \xRightarrow{\langle 0r_k, 1x_{n-1}\rangle} & (c_1r_1 \cdots c_{k-1}r_{k-1}, r_kx_1 \cdots x_{n-2}) \\ & \text{if } c_k = 0 \\ \xRightarrow{\langle 1r_k, 0x_1\rangle} & (c_1r_1 \cdots c_{k-1}r_{k-1}, x_2 \cdots x_{n-1}r_k) \\ & \text{if } c_k = 1 \end{array} \right.$$

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

The routing direction is given as follows:

1. If output $c_k r_k$ is available and $k = 1$, the cell has reached its destination; output the cell before the next shuffle if $c = 1$, and after the next unshuffle if $c = 0$.
2. If output $c_k r_k$ is available and $k > 1$, remove the two least significant bits from the routing tag and send the cell to the next stage.
3. If output $c_k r_k$ is unavailable and $k < n$, choose any other available outputs, attach the corresponding 2 bits for error-correcting to the routing tag and send the cell 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 8.18 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 $0x$. In either case, the error-correcting tag is just the second component $\bar{c}x$ in the link label $\langle cr, \bar{c}x\rangle$, where $x = x_{c+\bar{c}(n-1)}$, which is either x_1 or x_{n-1} depending on $c = 1$ (SN) or $c = 0$ (USN). Therefore, a cell deflected to link $\langle cr, \bar{c}x\rangle$ will return to its previous state via link $\langle \bar{c}x, cr\rangle$ in the next stage. This point is illustrated in Figure 8.19 by the same example given in Figure 8.16.

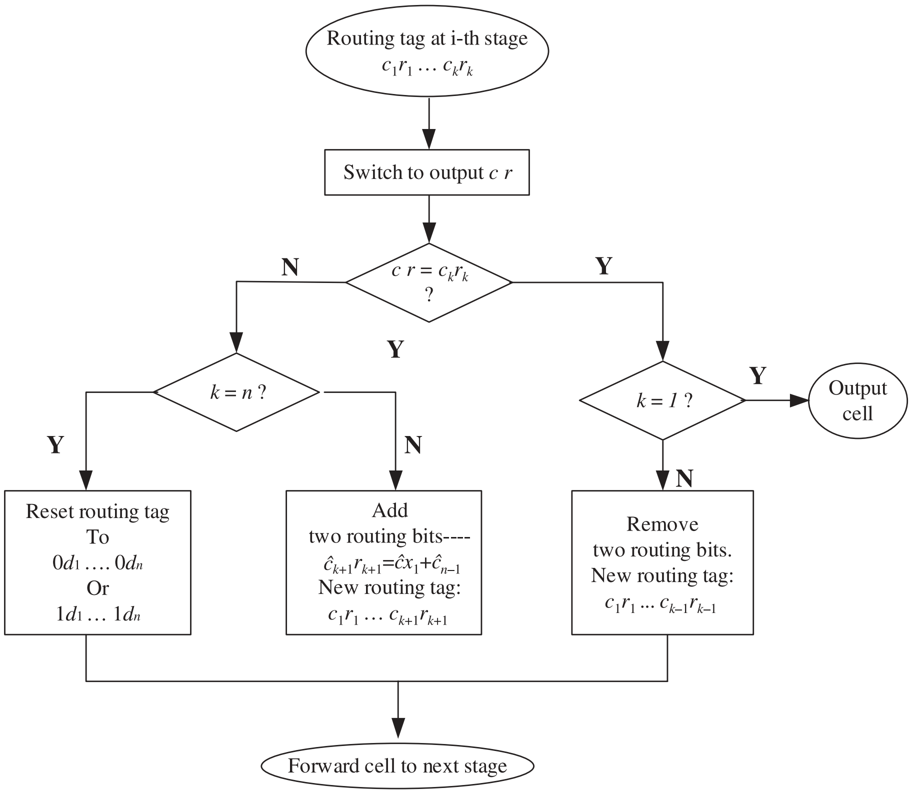
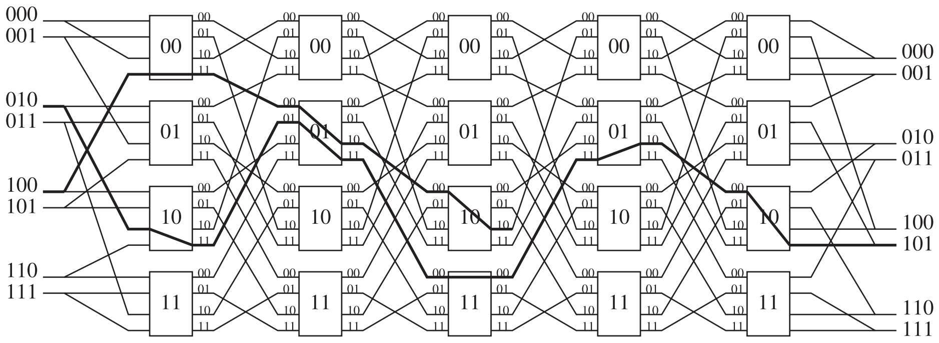

Routing tag at i-th stage
$c_1r_1 \ldots c_kr_k$
Switch to output $c\ r$
$c\ r = c_kr_k$?
N
Y
$k = n$?
Y
N
Y
$k = 1$?
Y
Output cell
N
Reset routing tag To $0d_1 \ldots 0d_n$ Or $1d_1 \ldots 1d_n$
Add two routing bits---- $\hat{c}_{k+1}r_{k+1} = \hat{c}x_1 + \hat{c}_{n-1}$ New routing tag: $c_1r_1 \ldots c_{k+1}r_{k+1}$
Remove two routing bits. New routing tag: $c_1r_1 \ldots c_{k-1}r_{k-1}$
Forward cell to next stage

Figure 8.18 Error-correcting routing algorithm.

This algorithm can implicitly handle successive deflections as shown by the finite-state machine representation of the algorithm in Figure 8.20. The state transitions when deflection occurred are given by

$$(c_1r_1\cdots c_kr_k, x_1\cdots x_{n-1}) \longrightarrow \begin{cases} \xRightarrow{\langle 0r,\,1x_{n-1}\rangle} & (c_1r_1\cdots c_kr_k1x_{n-1}, rx_1\cdots x_{n-2}) \\ & \text{if } c_kr_k \neq 0r \\ \xRightarrow{\langle 1r,\,0x_1\rangle} & (c_1r_1\cdots c_kr_k0x_1, x_2\cdots x_{n-1}r) \\ & \text{if } c_kr_k \neq 1r \end{cases}$$



State transition of cells A and B

A: 010 → (111011, 10) → (1110, 01) $\xrightarrow{\text{error}}$ (111000, 11) $\xrightarrow{\text{error correction}}$ (1110, 01) → (11, 10) → 101

B: 100 → (101011, 00) → (1010, 01) → (10, 10) → 100

Figure 8.19 Example of error-correcting routing in DSN.

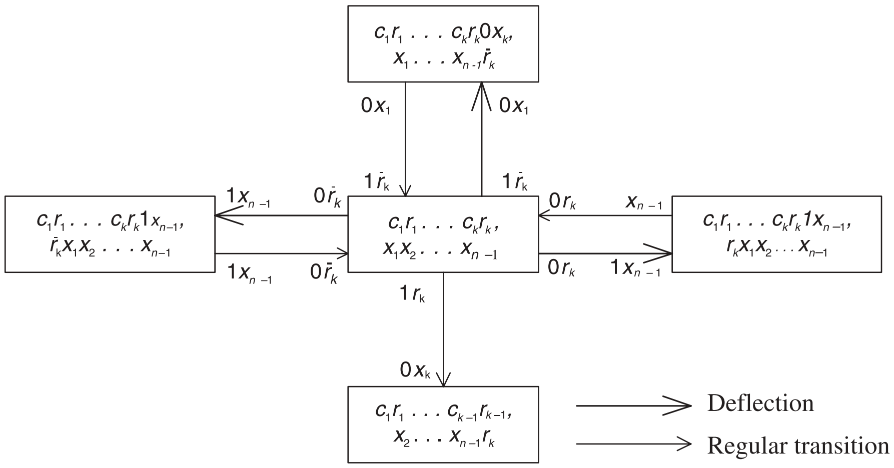

$c_1r_1 \ldots c_kr_k0x_k$, $x_1 \ldots x_{n-1}\bar{r}_k$
$c_1r_1 \ldots c_kr_k1x_{n-1}$, $\bar{r}_kx_1x_2 \ldots x_{n-1}$
$c_1r_1 \ldots c_kr_k$, $x_1x_2 \ldots x_{n-1}$
$c_1r_1 \ldots c_kr_k1x_{n-1}$, $r_kx_1x_2 \ldots x_{n-1}$
$c_1r_1 \ldots c_{k-1}r_{k-1}$, $x_2 \ldots x_{n-1}r_k$
$0x_1$
$0x_1$
$1\bar{r}_k$
$1\bar{r}_k$
$1x_{n-1}$
$0\bar{r}_k$
$1x_{n-1}$
$0\bar{r}_k$
$0r_k$
x_{n-1}
$0r_k$
$1x_{n-1}$
$1r_k$
$0x_k$
Deflection
Regular transition

Figure 8.20 Finite-state machine representation of the error-correcting routing algorithm when $c_k = 1$.

8.6 MULTICAST COPY NETWORKS

Figure 8.21 illustrates a serial combination of a copy network and a point-to-point switch for supporting point-to-multipoint communications [12]. The copy network replicates cells from various inputs simultaneously, and then copies of broadcast cells are routed to the final destination by the point-to-point switch.

A copy network consists of the following components and its basic structure is shown in Figure 8.22.

- Running adder network (RAN) generates running sums of the copy numbers, specified in the headers of input cells.
- Dummy address encoder (DAE) takes adjacent running sums to form a new header for each cell.

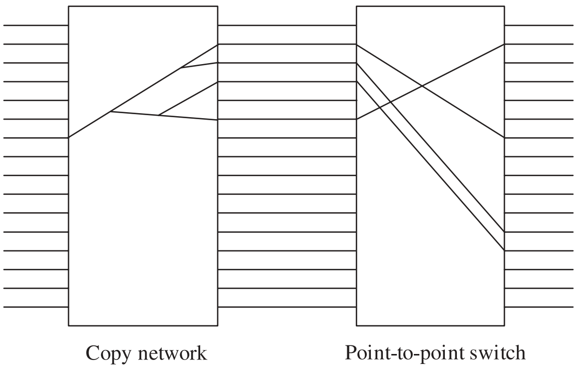


Figure 8.21 Multicast cell switch consists of a copy network and a point-to-point switch.

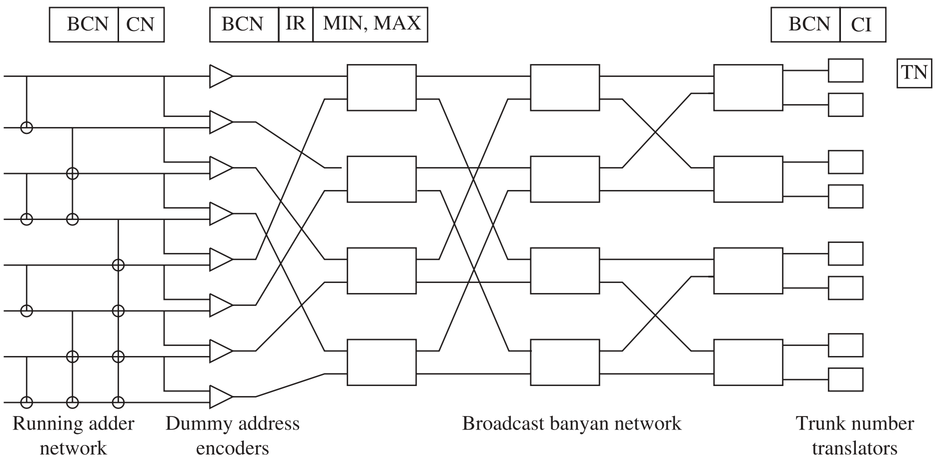


Figure 8.22 Basic components in a nonblocking copy network.

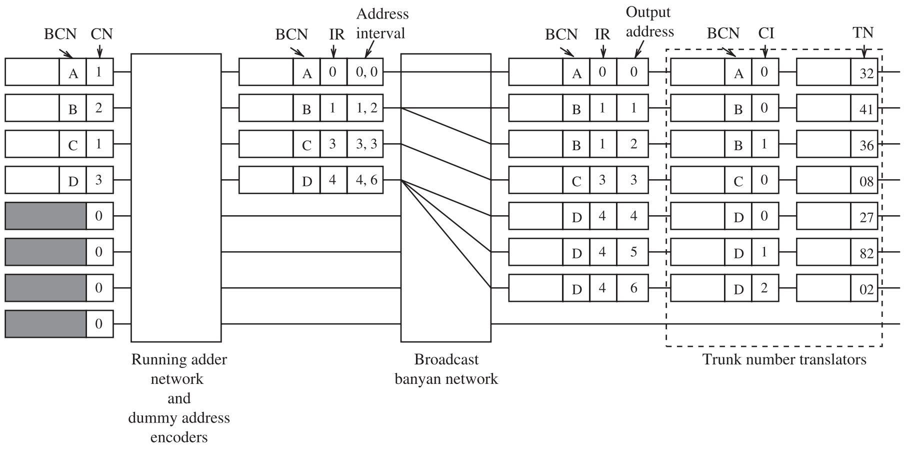


Figure 8.23 Header translations in the copy network.

- Broadcast banyan network (BBN) is a banyan network with broadcast switch nodes capable of replicating cells based on two-bit header information.
- Trunk number translator (TNT) determines the outgoing trunk number for each cell copy.

The multicasting mechanism of the copy network lies in the header translations illustrated in Figure 8.23. First, the number of copies (CNs) specified in the cell headers are added up recursively over the running adder network. Based on the resulting sums, the dummy address encoders form new headers with two fields: a dummy address interval and an index reference (IR). The dummy address interval is formed by the adjacent running sums, namely, the minimum (MIN) and the maximum (MAX). The index reference is set equal to the minimum of the address interval, and is used later by the trunk number translators to determine the copy index (CI). The broadcast banyan network replicates cells according to a Boolean interval splitting algorithm based on the address interval in the new header. When a copy finally appears at the desired output, the TNT computes its copy index from the output address and the index reference. The broadcast channel number (BCN) and the copy index form a unique identifier into a trunk number (TN), which is added to the cell header and used to route the cell to its final destination.

8.6.1 Broadcast Banyan Network

Generalized Self-Routing Algorithm. A broadcast banyan network is a banyan network with switch nodes that are capable of replicating cells. A cell arriving at each node can be either routed to one of the output links, or it can be replicated and sent out on both links. There are three possibilities and the uncertainty of making a decision is $\log_2 3 = 1.585$, which means that the minimum header information for a node is 2 bits.

Figure 8.24 illustrates a generalization of the 1-bit self-routing algorithm to the multi-bit case for a set of arbitrary N-bit destination addresses. When a cell arrives at a node in stage k, the cell routing is determined by the kth bits of all destination addresses in the header. If they are all '0' or all '1', then the cell will be sent out on link 0 or link 1, respectively. Otherwise, the cell and its copy are sent out on both links and the destination addresses in

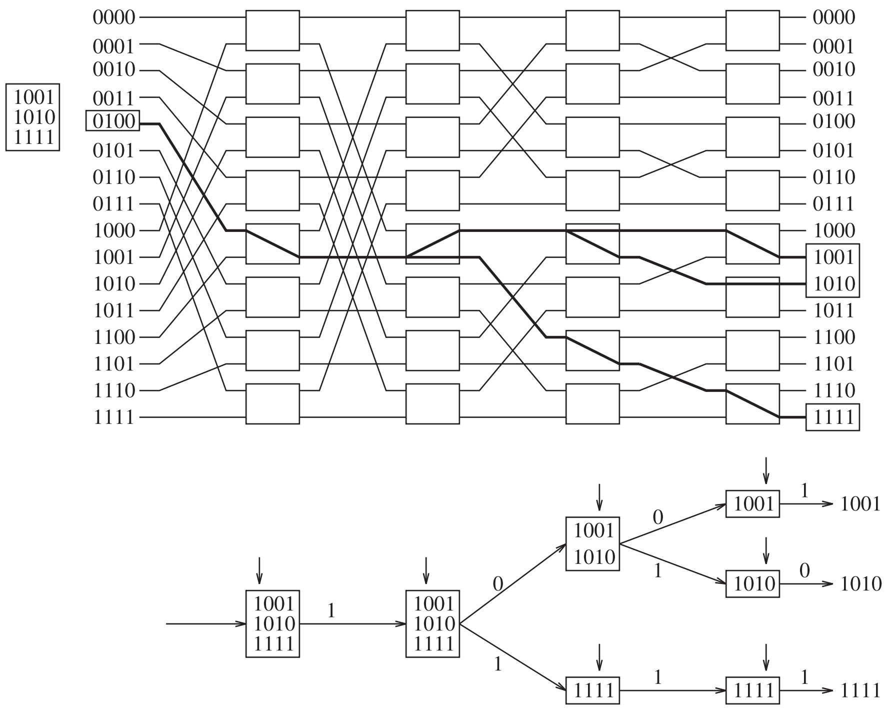


Figure 8.24 Input–output tree generated by generalized self-routing algorithm.

the header are modified correspondingly to the two cell copies: the header of the cell copy sent out on link 0 or link 1, contains those addresses in the original header with the kth bit equal to 0 or 1, respectively.

Several problems may arise in implementing the generalized self-routing algorithm. First, a cell header contains a variable number of addresses and the switch nodes have to read all of them. Second, the cell header modification depends on the entire set of addresses, which is a processing burden on switch nodes. Finally, the set of paths from any input to a set of outputs forms a tree in the network. The trees generated by an arbitrary set of input cells are not link-independent in general and the network is obviously blocking due to the irregularity of the set of actual destination addresses in the header of each cell. However, fictitious addresses instead of actual addresses can be used in the copy network, where cells are replicated but need not be routed to the actual destinations. The fictitious addresses for each cell may then be arranged to be contiguous such that an address interval consisting of the MIN and the MAX can represent the whole set of fictitious addresses. The address intervals of input cells can be specified to be monotonic to satisfy the nonblocking condition for the broadcast banyan network described below.

Boolean Interval Splitting Algorithm. An address interval is a set of contiguous N-bit binary numbers, which can be represented by two numbers, namely, the minimum and the maximum. Suppose that a node at stage k receives a cell with the header containing an address interval specified by the two binary numbers: $\min(k-1) = m_1 \cdots m_N$ and $\max(k-1) = M_1 \cdots M_N$, where the argument $(k-1)$ denotes stage $(k-1)$ from where

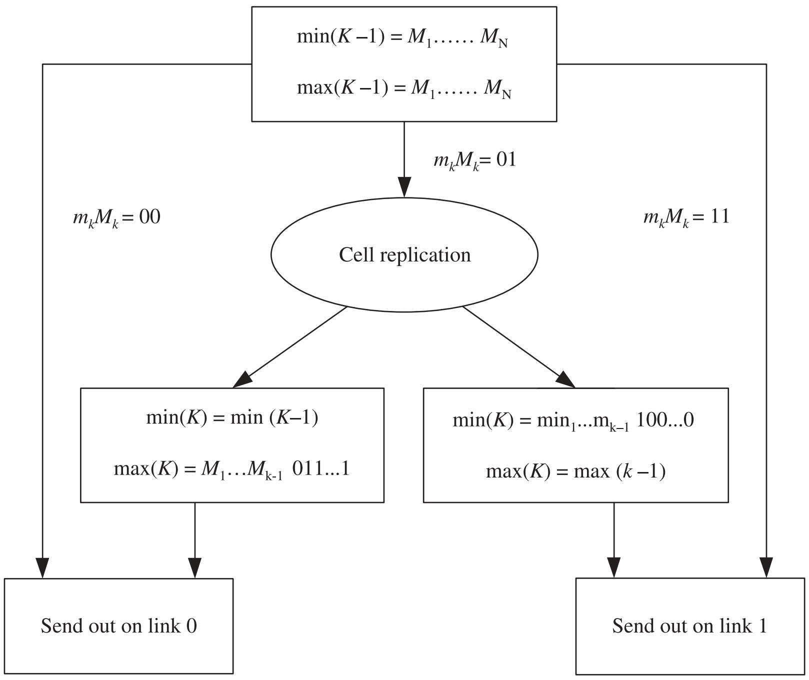


Figure 8.25 Switch node logic at stage k of a broadcast banyan network.

the cell came to stage k. The generalized self-routing algorithm gives the direction for cell routing as follows, and as illustrated in Figure 8.25.

- If $m_k = M_k = 0$ or $m_k = M_k = 1$, then send the cell out on link 0 or 1, respectively.
- If $m_k = 0$ and $M_k = 1$, then replicate the cell, modify the headers of both copies (according to the scheme described below) and send each copy out on the corresponding link.

The modification of a cell header is simply splitting the original address interval into two subintervals, as expressed in the following recursion. For the cell sent out on link 0,

$$\begin{cases} \min(k) = \min(k-1) = m_1 \cdots m_N \\ \max(k) = M_1 \cdots M_{k-1} 01 \cdots 1, \end{cases}$$

and for the cell sent out on link 1,

$$\begin{cases} \min(k) = m_1 \cdots m_{k-1} 10 \cdots 0 \\ \max(k) = \max(k-1) = M_1 \cdots M_N. \end{cases}$$

Figure 8.26 illustrates the Boolean interval splitting algorithm. From the rules it is realized that $m_i = M_i$, $i = 1, \ldots, k-1$ holds for every cell that arrives at stage k. The event $m_k = 1$ and $M_k = 0$ will never occur.

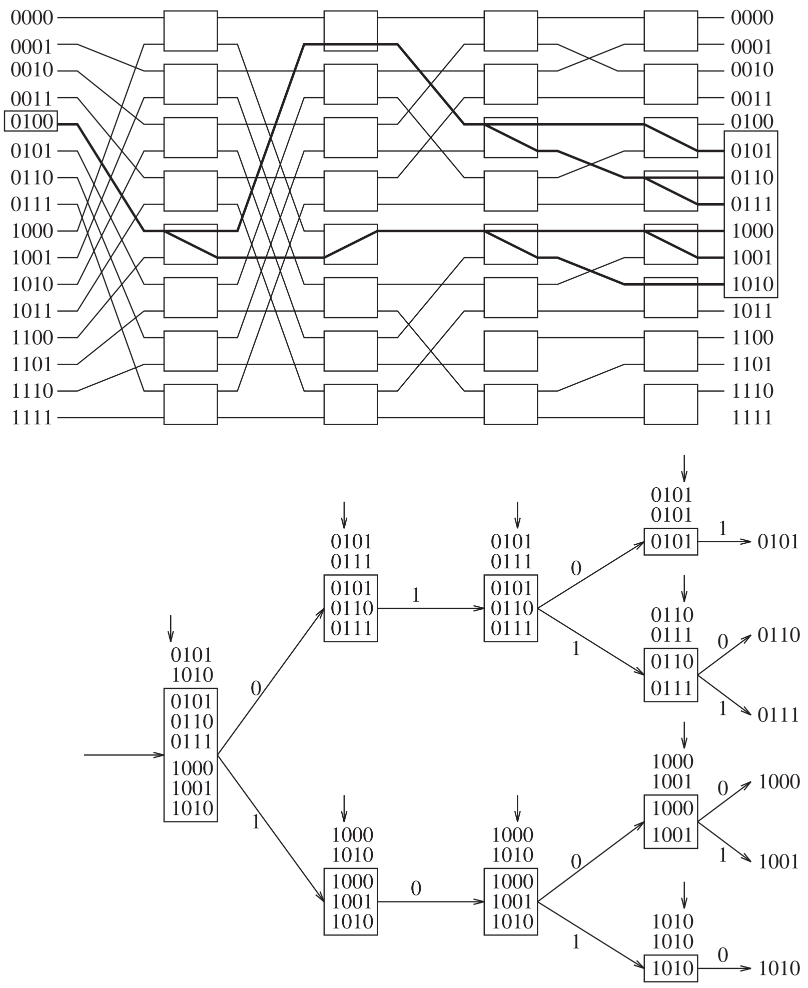


Figure 8.26 Boolean interval splitting algorithm generates the tree while replicating a cell according to the address intervals.

Nonblocking Condition of Broadcast Banyan Networks. A broadcast banyan network is nonblocking if the active inputs $x_1, \ldots, x_k$ and the corresponding sets of outputs $Y_1, \ldots, Y_k$ satisfy the following:

(1) (Monotone): $Y_1 < Y_2 < \cdots < Y_k$, or $Y_1 > Y_2 > \cdots > Y_k$.

(2) (Concentration): Any input between two active inputs is also active.

The above inequality $Y_i < Y_j$ means that every output address in Y_i is less than any output address in Y_j. Figure 8.27 illustrates a nonblocking example with active inputs $x_1 = 7$, $x_2 = 8$, $x_3 = 9$, and corresponding outputs $Y_1 = \{1, 3\}$, $Y_2 = \{4, 5, 6\}$, $Y_3 = \{7, 8, 10, 13, 14\}$.

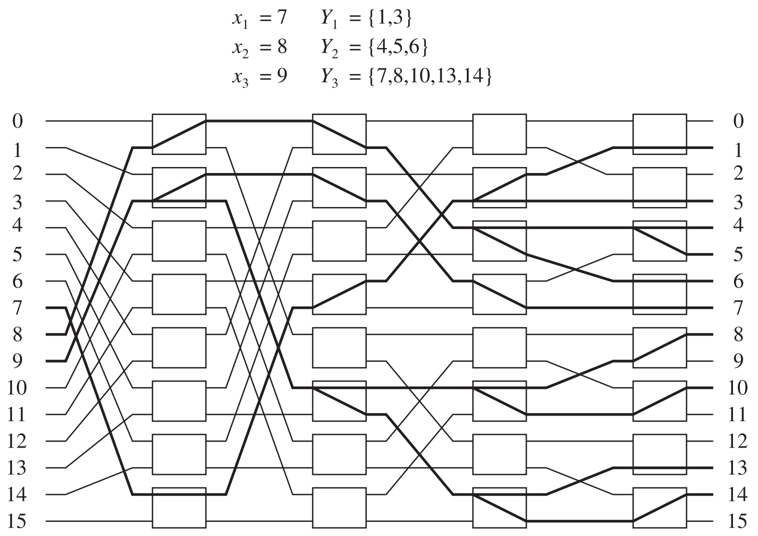

$x_1 = 7$ $Y_1 = \{1,3\}$
$x_2 = 8$ $Y_2 = \{4,5,6\}$
$x_3 = 9$ $Y_3 = \{7,8,10,13,14\}$
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Figure 8.27 Example to demonstrate the nonblocking condition of broadcast banyan network.

8.6.2 Encoding Process

The RAN together with the DAEs is used to arrange the destination addresses for each cell so that every eligible cell can be replicated appropriately in the broadcast banyan network without any conflicts. Cell replications in the broadcast banyan network are aided by two processes, an encoding process and a decoding process. The encoding process transforms the set of copy numbers, specified in the headers of incoming cells, into a set of monotone address intervals that form the cell headers in the broadcast banyan network. This process is carried out by a running adder network and a set of dummy address encoders. The decoding process determines the destinations of copies with the trunk number translators.

The recursive structure of the $\log_2 N$-stage running adder network is illustrated in Figure 8.28. The adder network consists of $(N/2)\log_2 N$ adders, each with two inputs and two outputs where a vertical line denotes a pass. The east output is the sum of both the west and the north inputs, while the south output just propagates the north input down. The running sums of CNs are then generated at each port after $\log_2 N$ stages, before the dummy address encoders form the new headers from adjacent running sums. The new header consists of two fields: one is the dummy address interval represented by two $\log_2 N$-bit binary numbers (the minimum and the maximum), and the other contains an index reference that is equal to the minimum of the address interval. Note that the length of each interval is equal to the corresponding copy number in both addressing schemes.

Denote S_i as the ith running sum, the sequence of dummy address intervals will be generated as follows:

$$(0, S_0 - 1), (S_0, S_1 - 1), \ldots, (S_{N-2}, S_{N-1} - 1),$$

where the address is allocated beginning with 0. As shown in the previous section, this sequence satisfies the nonblocking condition over the broadcast banyan network.

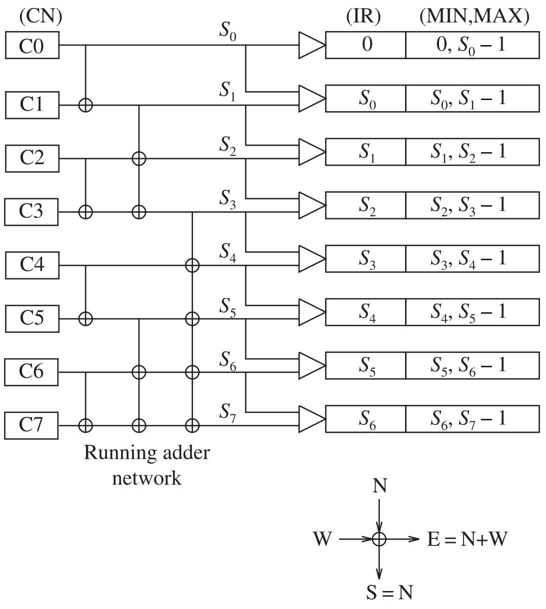


Figure 8.28 Running adder network and dummy address encoders.

8.6.3 Concentration

To satisfy the nonblocking condition of the broadcast banyan network, idle inputs between active inputs must be eliminated. This function should be performed before cells enter the broadcast banyan network, for example, prior to the BBN or right after the DAE in Figure 8.22. A reverse banyan network is thus used to concentrate active inputs into a contiguous list. As illustrated in Figure 8.29, the routing address in the reverse banyan network is determined by the running sums over activity bits to produce a set of continuous monotonic addresses.

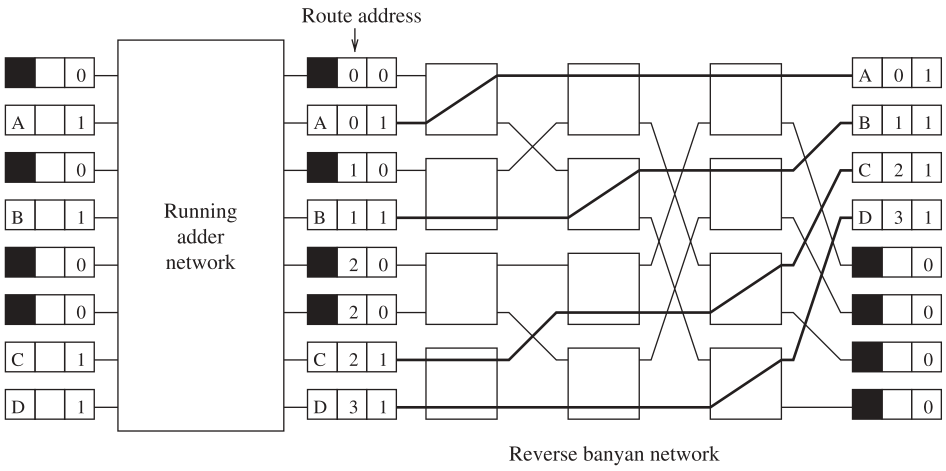


Figure 8.29 Input concentrator consists of a running adder network and a reverse banyan network.

8.6.4 Decoding Process

When a cell emerges from the broadcast banyan network, the address interval in its header contains only one address, that is, according to Boolean interval splitting algorithm,

$$\min(\log_2 N) = \max(\log_2 N) = \text{output address}.$$

The cell copies belonging to the same broadcast channel should be distinguished by copy index, which is determined at the output of broadcast banyan network (see Figure 8.30) by,

$$\text{copy index} = \text{output address} - \text{index reference}.$$

Recall that the index reference is initially set equal to the minimum of address interval.

A trunk number translator is used to assign the actual address to each cell copy so that it will be routed to its final destination in the succeeding point-to-point switch. Trunk number assignment can be accomplished by a simple table lookup identifier (searching key) that consists of the broadcast channel number (BCN) and the copy index (CI) associated with each cell. When a TNT receives a cell copy, it first converts the output address and IR into CI, and then replaces the BCN and CI with the corresponding trunk number in the translation table. The translation process is illustrated in Figure 8.31.

8.6.5 Overflow and Call Splitting

Overflow will occur in the RAN of the copy network when the total number of copy requests exceeds the capacity of the copy network. If partial service (also called call splitting) is not allowed in cell replication and a cell must generate all its copies in a time slot, then the throughput may be degraded when overflow occurs. As illustrated in Figure 8.32, overflow occurs at port 3 and only five cell copies are allowed although more than eight requests are available.

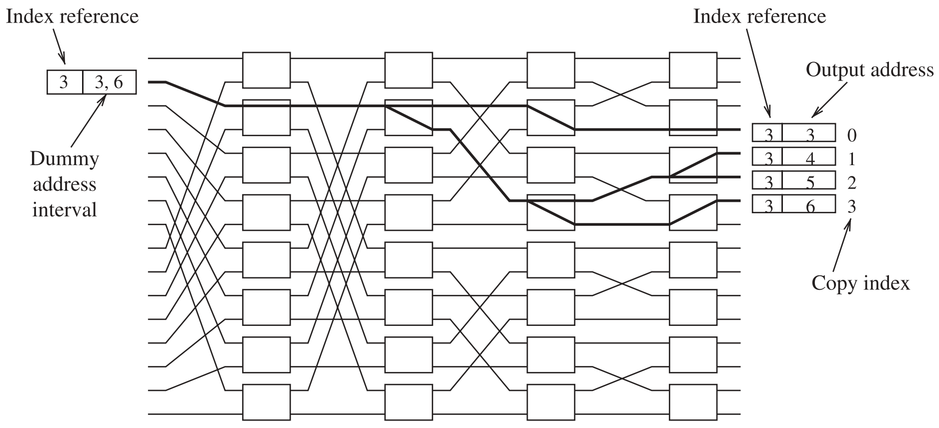


Figure 8.30 Computation of copy indexes.

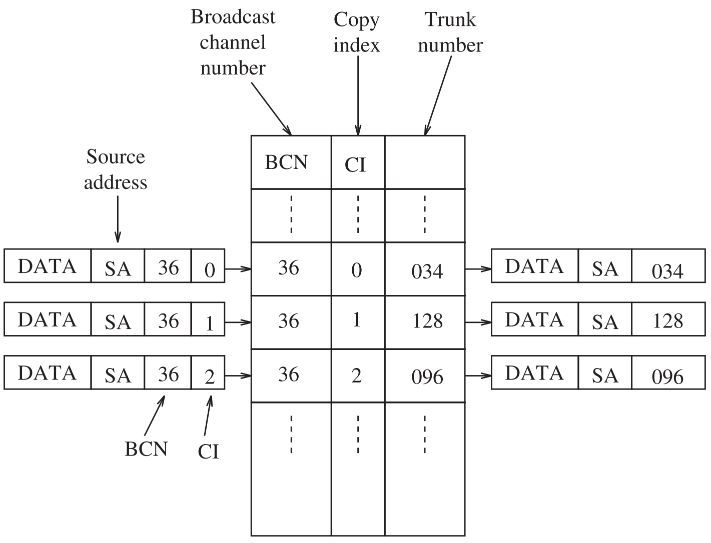


Figure 8.31 Trunk number translation by table lookup.

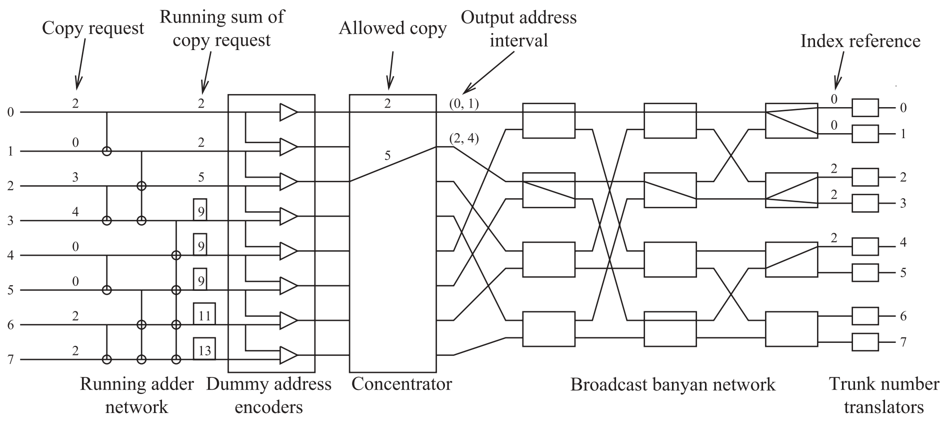


Figure 8.32 8 × 8 nonblocking copy network without call-splitting: only five instead of eight cell copies are allowed in this time slot.

8.6.6 Overflow and Input Fairness

Overflow will also introduce unfairness among incoming cells, because the starting point of the RAN is fixed. Since the calculation of running sum always starts from input port 0 in every time slot, lower numbered input ports have higher service priorities than the higher numbered ports.

This unfairness problem will be solved if the RAN is re-designed to calculate the running sums cyclically starting from any input port, and the starting point of computing the running sums in every time slot is determined adaptively by the overflow condition in the previous time slot. A cyclic RAN (CRAN) is illustrated in Figure 8.33. The current starting point is port 3, and call-splitting is performed at port 6 and the new starting point in the next slot

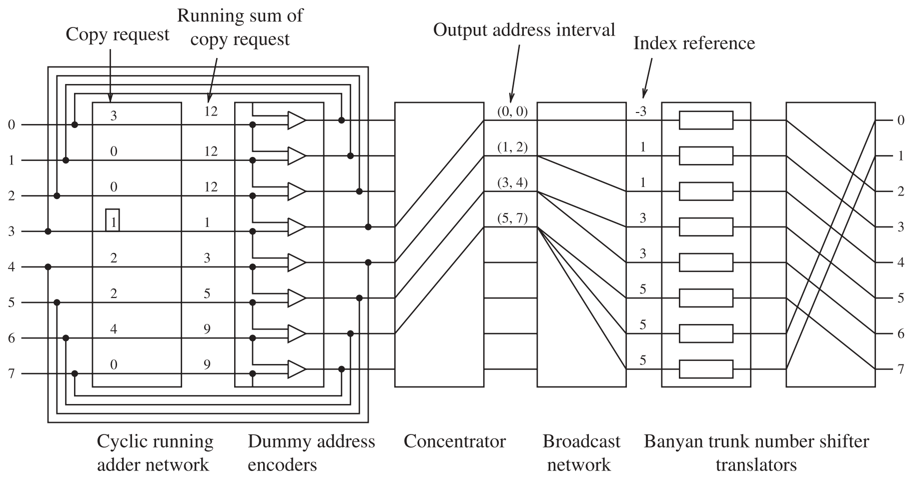


Figure 8.33 CRAN in an 8 × 8 copy network.

will be port 6. The negative index reference −3, provided by the DAE, implies that the copy request from port 3 is a residual one, and three copies have been generated in the previous time slot.

Cyclic running adder network Figure 8.34 shows the structure of an 8 × 8 CRAN. The associated cell header format consists of three fields: starting indicator (SI), running sum (RS), and routing address (RA). Only one port, the starting point, has a nonzero SI

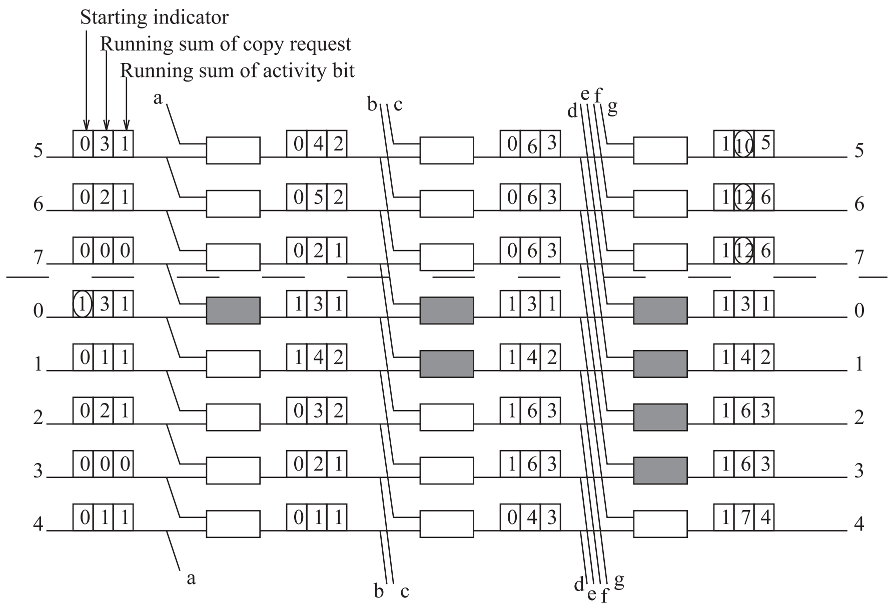


Figure 8.34 8 × 8 cyclic running adder network (CRAN).

initially. The RS field is initially set equal to the number of copies requested by the input cell. The RA field is initially set equal to 1 if the port is active, otherwise it is set to 0. At the output of the running adder network, the RA field will carry the running sum over activity bits, to be used as the routing address in the following concentrator.

A set of cyclic passing paths is implemented in each stage of CRAN so that recursive computation of running sums can be done cyclically. In order to emulate the actual running sums computation from a starting point, however, some passing paths should be virtually cut as illustrated in Figure 8.34, which is equivalent to the shaded nodes ignoring their links while computing the running sums. These nodes are preceded by a cell header with the SI field equal to 1, as it is propagated from the starting point over the CRAN. The header modification in a node is summarized in Figure 8.35.

The next starting point will remain the same unless overflow occurs, in which case, the first port facing the overflow will be the next starting point. If we denote the starting point as port 0, and number other ports from 1 to $N - 1$ in a cyclic manner, then the SI bit that indicates the next starting point is updated with adjacent RS fields at each port as follows:

$$SI_0 = \begin{cases} 1 & \text{if } RS_{N-1} \le N \\ 0 & \text{otherwise} \end{cases}$$

and

$$SI_i = \begin{cases} 1 & \text{if } RS_{i-1} \le N \text{ and } RS_i > N \\ 0 & \text{otherwise,} \end{cases}$$

where $i = 1, 2, \ldots, N - 1$.

In order to support call-splitting, every input port should know how many copies are served in each time slot. This piece of information is called starting copy number (SCN). A set of feedback loops is then established to send this information back to input ports after it is determined with adjacent running sums as follows:

$$SCN_0 = RS_0,$$

and

$$SCN_i = \begin{cases} \min(N - RS_{i-1}, RS_i - RS_{i-1}) & \text{if } RS_{i-1} < N \\ 0 & \text{otherwise} \end{cases}$$

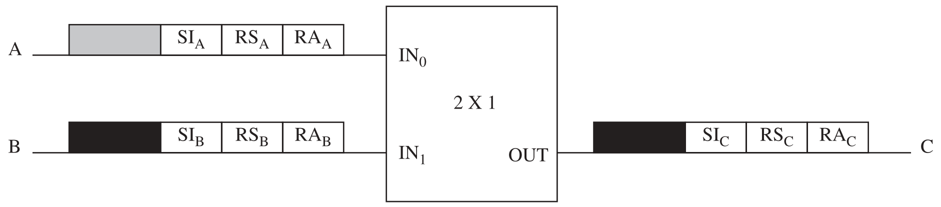


Figure 8.35 Operation of a node in CRAN.

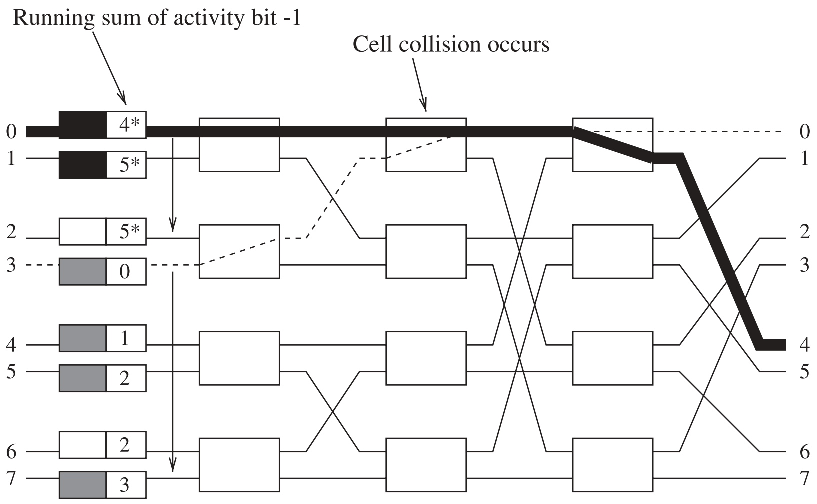


Figure 8.36 Cyclic monotone addresses give rise to cell collisions in the reverse banyan network. Port 2 and port 6 are idle.

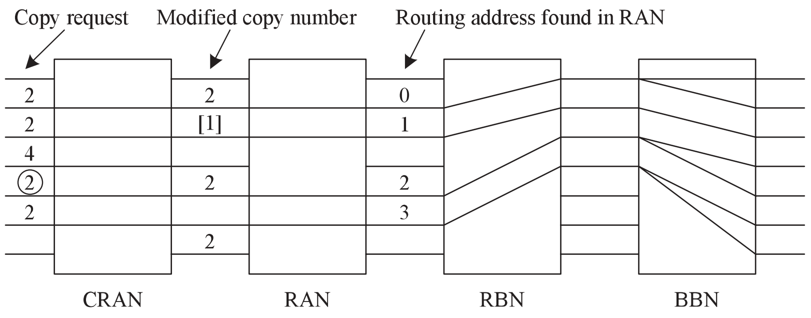


Figure 8.37 Additional RAN is used to concentrate active cells. The starting point is marked by encircling its copy request.

Concentration. The starting point in a CRAN may not be port 0, and the resulting sequence of routing addresses in the reverse banyan network (RBN) may not be continuous monotone. As illustrated in Figure 8.36, internal collisions may occur in the RBN.

This problem can be solved if an additional RAN with fixed starting point (port 0) is added in front of the RBN. As shown in Figure 8.37, this additional RAN will recalculate the running sums of RAs so that the resulting sequence of RAs becomes continuous monotone.

REFERENCES

[1] C. Clos, "A study of non-blocking switching network," *Bell System Technical Journal*, vol. 32, no. 2, pp. 404–426 (Mar. 1953).

[2] V. E. Benes, "Optimal rearrangeable multistage connecting networks," *Bell System Technical Journal*, vol. 43, pp. 1641–1656 (July 1964).

[3] J. S. Turner and L. F. Wyatt, “A packet network architecture for integrated services,” in *Proc. Globecom’83*, San Diego, California, pp. 2.1.1–2.1.6 (Nov. 1983).

[4] J. J. Kulzer and W. A. Montgomery, “Statistical switching architecture for future services,” in *Proc. ISS’84*, Florence, Italy, pp. 22A.4.1–22A.4.6 (May 1984).

[5] J. S. Turner, “Design of a broadcast packet switching network,” *IEEE Transaction on Communications*, vol. 36, pp. 734–743 (June 1998).

[6] L. R. Goke and G. J. Lipovski, “Banyan networks for partitioning multiprocessor systems,” in *Proc. 1st Annu. Int. Symp. Comput. Architecture*, Gainesville, Florida, pp. 21–28 (Dec. 1973).

[7] K. E. Batcher, “Sorting networks and their application,” in *Proc. Spring Joint Comput. Conf. AFIPS*, Washington, DC, pp. 307–314 (1968).

[8] B. Bingham and H. Bussey, “Reservation-based contention resolution mechanism for Batcher-banyan packet switches,” *Electronic Letters*, vol. 24, no. 13, pp. 772–773 (June 1988).

[9] J. N. Giacopelli, J. J. Hickey, W. S. Marcus, W. D. Sincoskie, and M. Littlewood, “Sunshine: a high-performance self-routing broadband packet switch architecture,” *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1289–1298 (Oct. 1991).

[10] F. A. Tobagi and T. Kwok, “The tandem banyan switching fabric: a simple high-performance fast packet switch,” in *Proc. IEEE INFOCOM’91*, Bal Harbour, Florida, pp. 1245–1253 (Apr. 1991).

[11] S. C. Liew and T. T. Lee, “$N \log N$ dual shuffle-exchange network with error-correcting routing,” *IEEE Transactions on Communications*, vol. 42, no. 2, pp. 754–766 (Feb. 1994).

[12] T. T. Lee, “Nonblocking copy networks for multicast packet switching,” *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1455–1467 (Dec. 1988).