

# CLOS-NETWORK SWITCHES

---

In this chapter we consider an approach to building modular switches. The architecture is based on the Clos network (see Figure 12.1). Switch modules are arranged in three stages, and every module is interconnected with every other module in the adjacent stage via a unique link. Here, the three stages are referred to as the input stage, middle stage, and output stage. The modules in those stages are accordingly called input modules (IMs), central modules (CMs), and output modules (OMs). Each module is assumed to be non-blocking and could be, for example, the crossbar switches. Inputs are partitioned into groups of the same size, and the inputs in the same group are connected to an input module. Let  $n$  be the number of inputs per group, and  $k$  the number of input modules. The total number of inputs is given by  $N = n \times k$ . On the other hand, a mirror structure can be found on the output side. In the middle, there are  $m k \times k$  central modules, as each link on a central module is dedicated to either an IM or OM.

One could argue for only considering the two-stage interconnection network, in which every pair of modules of adjacent stages are interconnected with a dedicated link. In this case, only one cell can be transmitted between any pair of modules because there is just one path between them, causing a very high blocking probability. In a Clos network, however, two cells from an input module can take distinct paths via different central modules to get to the same output module. The central modules in the middle stage can be viewed as the routing resources shared by all input and output modules. One can expect that this will give a better tradeoff between the switch performance and complexity.

Because of this property, the Clos network was widely adopted in the traditional circuit-switched telephone network where a path is reserved in the network for each call. If a Clos network has enough central modules, a path can always be found for any call between an idle input and an idle output. Such a property is called nonblocking. Basically, there are two senses of nonblocking, strictly and rearrangeably. In a strictly nonblocking Clos network,

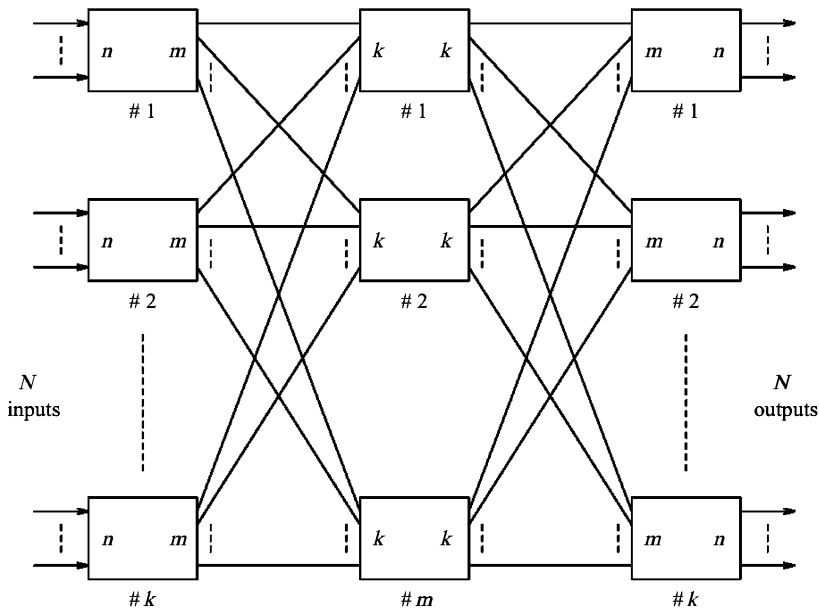


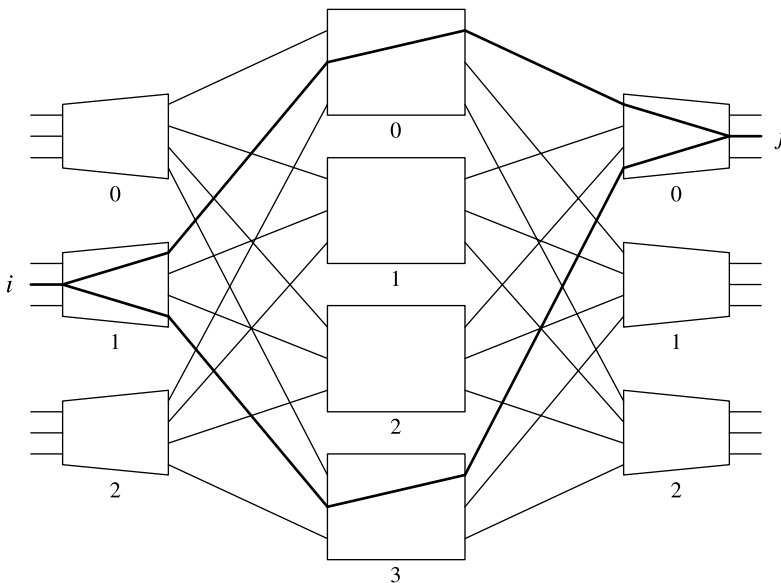
Figure 12.1 A Clos-network switch.

every newly arriving call will find a path across the network without affecting the existing calls. With rearrangeably nonblocking, we may have to arrange paths for some existing calls in order to accommodate a new arrival.

This chapter will now focus on how the Clos network is used in packet switching. The rest of the chapter is organized as follows: first, we describe the basic routing properties in a Clos network and formulate the routing as a scheduling problem. Second, we discuss several scheduling schemes for the Clos network switch and classify them into three categories. First category is sequential optimal matching, which gives the maximum input–output matching at every timeslot at the cost of high complexity. In this category we also discuss two algorithms: The looping algorithm and  $m$ -matching algorithm [1]. The second category is semi-parallel matching, which provides the balance between performance and complexity, and we illustrate one algorithm, namely, the Euler partition algorithm [2]. The third category is parallel heuristic matching, which may not give the best matching but enjoys low complexity. We describe six algorithms: Karol’s algorithm [3], the frame-base matching algorithm for the Clos network (f-MAC) [4], the concurrent matching algorithm for the Clos network (c-MAC) [4], the dual-level matching algorithm for the Clos network (d-MAC) [4], the random dispatching algorithm (Atlanta switch) [3], and the concurrent round-robin dispatching (CRRD) algorithm [3, 5].

## 12.1 ROUTING PROPERTY OF CLOS NETWORK SWITCHES

For the  $N \times N$  switch shown in Figure 12.1, both the inputs and the outputs are divided into  $k$  modules with  $n$  lines each. The dimensions of the input and output modules are  $n \times m$  and  $m \times n$ , respectively, and there are  $m$  middle-stage modules, each of size  $k \times k$ .



**Figure 12.2** Two possible routing paths from input  $i$  to input  $j$  in the Clos network.

As illustrated in Figure 12.2, the routing constraints of the Clos network are briefly stated as follows:

1. Any central module can only be assigned to one input of each input module, and one output of each output module.
2. Input  $i$  and output  $j$  can be connected through any central module.
3. The number of alternate paths between input  $i$  and output  $j$  is equal to the number of central modules.

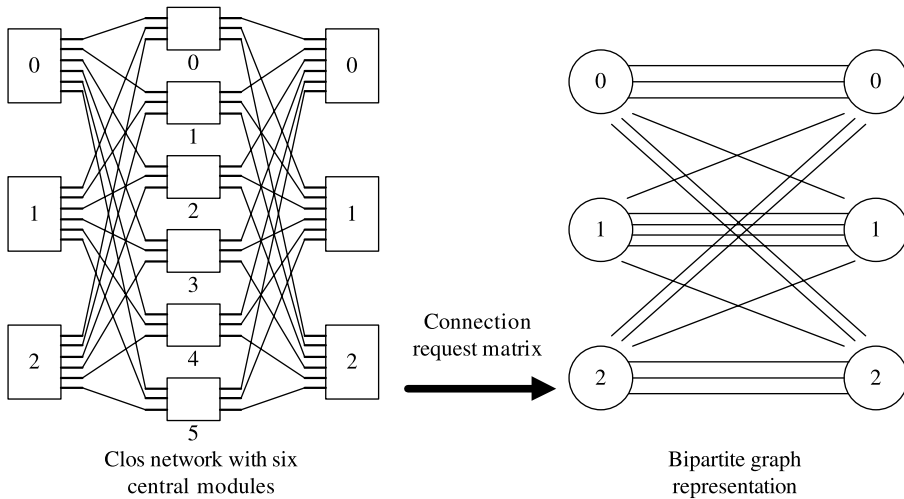
The routing problem is how to direct the input cells to the respective output modules without path conflicts. For every time slot, the traffic between input modules and output modules can be written as

$$T = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,k} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,k} \\ \cdots & \cdots & \cdots & \cdots \\ t_{k,1} & t_{k,2} & \cdots & t_{k,k} \end{pmatrix}$$

where  $t_{ij}$  represents the number of cells arriving at the  $i$ th input module destined for the  $j$ th output module. The row sum is the total number of cells arriving at each input module, while the column sum is the number of cells destined for the output module, and they are denoted as

$$R_i = \sum_{j=1}^k t_{i,j} \leq n \leq m, \quad i = 1, 2, \dots, k, \tag{12.1}$$

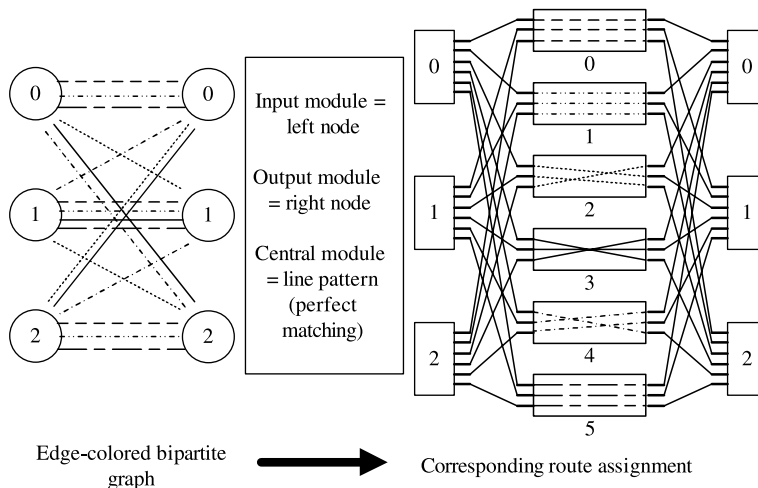
$$S_j = \sum_{i=1}^k t_{i,j} \leq n, \quad j = 1, 2, \dots, k. \tag{12.2}$$



**Figure 12.3** Bipartite graph representation of the Clos network routing problem.

The routing problem in the Clos network can be formulated as an edge coloring problem in a bipartite graph. With reference to Figure 12.3, a Clos network switch with six center modules and a given connection matrix can be transformed into a bipartite graph representation, where the numbers of lines connecting an input module and an output module is the number of requesting connections.

With the bipartite graph shown in Figure 12.3, the routing problem in the Clos network is transformed into an edge coloring problem in the graph. The aim is to use the minimum number of colors to color those lines in the bipartite graph such that there are no two colored lines the same for each module. Figure 12.4 is the solution of the edge coloring problem for the bipartite graph shown in Figure 12.3. Note that the lines with the same color indicate the routing in the Clos network via the same center module.



**Figure 12.4** Edge coloring approach for the Clos network routing. Different edge colors are represented by different edge patterns.

Another way to solve the routing problem in the Clos network is matrix decomposition. This mathematical expression is to decompose the connection matrix into the summation of several sub-matrices

$$T = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,k} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,k} \\ \cdots & \cdots & \cdots & \cdots \\ t_{k,1} & t_{k,2} & \cdots & t_{k,k} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \cdots & \cdots & \cdots & \cdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} \end{pmatrix} + \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,k} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,k} \\ \cdots & \cdots & \cdots & \cdots \\ b_{k,1} & b_{k,2} & \cdots & b_{k,k} \end{pmatrix} + \cdots$$

And each of the sub-matrices must have the following property:

$$\sum_{j=1}^k a_{i,j} \leq 1, \quad \sum_{j=1}^k b_{i,j} \leq 1, \quad i = 1, 2, \dots, k. \tag{12.3}$$

and

$$\sum_{i=1}^k a_{i,j} \leq 1, \quad \sum_{i=1}^k b_{i,j} \leq 1, \quad j = 1, 2, \dots, k. \tag{12.4}$$

Figure 12.5 illustrates the matrix decomposition approach in solving the routing problem in the Clos network for the same example given earlier in Figure 12.3.

In summary, the routing problem in Clos network can be interpreted as three equivalent questions:

1. How to assign central routes in a Clos network with  $m$  central modules in order to accommodate a set of connection requests?

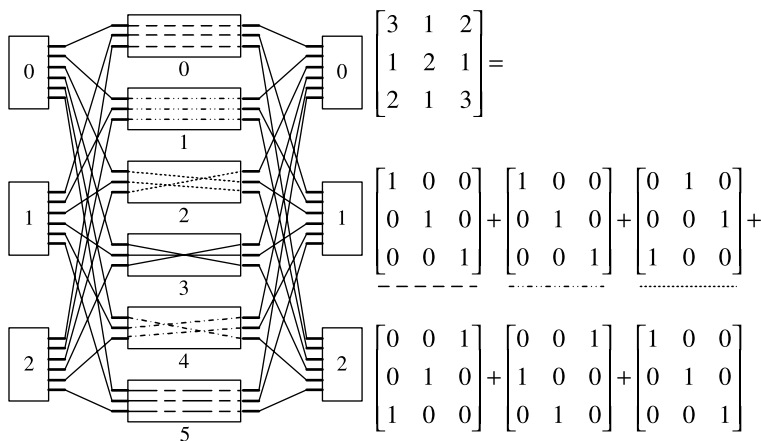


Figure 12.5 Matrix decomposition approach for Clos network routing.

2. How to edge-color a given bipartite graph with maximum degree  $m$ ?
3. How to decompose a matrix with row/column sum  $\leq m$  into  $m$  matrices with row/column sum  $\leq 1$ ?

In the following, we will discuss several well-known scheduling schemes for the Clos network in detail. They can be divided into three categories: (1) Sequential optimal matching; (2) Semi-parallel matching; (3) Parallel heuristic matching.

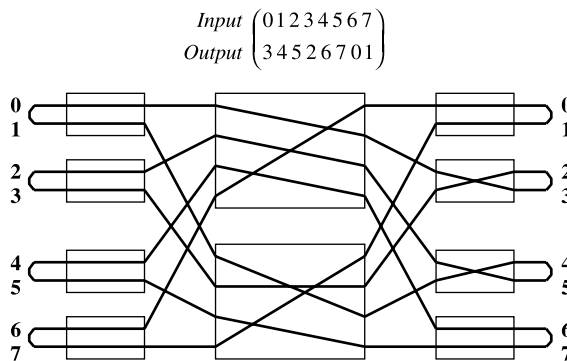
### 12.2 LOOPING ALGORITHM

Looping algorithm is a simple sequential optimal matching algorithm for the Clos network. The advantage of this algorithm is its simplicity and low complexity  $O(N)$ . Its disadvantage is that it only works for the  $m = 2$  Clos network. In other words, the looping algorithm is only suitable for the Clos network with two center modules and input/output modules with a size of  $2 \times 2$ .

For a given input traffic matrix, the looping algorithm works as follows:

- Step 1. Start the looping algorithm from any arbitrary input port.
- Step 2. Find the desired output module that contains the destined output port via one of the center modules.
- Step 3. Loop back from the same output module in step 2, but with the alternative output port, and trace to the corresponding input port via the alternative center module from step 2.
- Step 4. Loop back from the same input module found in step 3, but with the alternative input port. Continue with steps 2–4.
- Step 5. If a close loop or an idle input/output is found, restart the looping algorithm from step 1.

An example to better illustrate the looping algorithm is as follows. Assume a Clos network switch with a total size of  $8 \times 8$ , consisting of four input modules and four output modules, each of which has a size of  $2 \times 2$ , and two center modules, each of which has a size of  $4 \times 4$ . For a given set of input/output connection requests, the looping algorithm



**Figure 12.6** Looping algorithm in a  $8 \times 8$  Clos network.

gives two sets of matching in Figure 12.6, one of them is maximum matching, while the other is not necessary.

We can trace a few steps of this example. Let us start with input 0: it desires output 3 and reaches output 3 via center module 0. Next we loop back from output 2 to input 3 via center module 1. Then input 2 takes its turn and finds output 5 via center module 0. Finally, we loop back from output 4 to input 1 via center module 1. Now we have a closed loop. The looping algorithm may restart with any of the remaining unassigned input ports. Eventually, we will complete all possible matches between the inputs and outputs.

Notice that, in the looping algorithm, each input–output pair takes turns to assign routes via the center module. Therefore, the total complexity of this algorithm is  $O(N)$ .

### 12.3 *m*-MATCHING ALGORITHM

The *m*-matching is another sequential optimal matching algorithm for the Clos network. The *m*-matching algorithm uses the bipartite graph as an approach to find perfect matching (only if the bipartite graph is regular). Thus, it does not have the switch size constraint.

As mentioned in the earlier section, the edge coloring problem tries to use the minimum amount of colors to color the bipartite graph such that there are no two identical colors from the same edge. The *m*-matching algorithm solves this edge coloring problem by assigning one color at a time. It assigns one color to the matching and removes the corresponding edges from the bipartite graph. Then it continues this process till all the edges are colored. Each color may take  $O(N)$  steps to find a perfect matching from the bipartite graph. Thus with *m* different colors, the *m*-matching algorithm has a time complexity of  $O(mN)$ .

The drawback of this matching algorithm is that it is incapable of handling irregular bipartite graphs. An irregular bipartite graph is the one that does not have an equal number of degrees from each node. In other words, each input module of the Clos network does not have an equal number of requests. In the case of the irregular bipartite graph, one may use maximum-size matching to transform an irregular bipartite graph into a regular one at the cost of increased complexity of  $O(N^{2.5})$ .

### 12.4 EULER PARTITION ALGORITHM

Both the looping algorithm and *m*-matching algorithm can give perfect matching for the Clos network. In other words, they always provide the best possible matching between input and output for a given traffic matrix. However, the former has a switch size constraint and the latter suffers high computation complexity. Next, we discuss a semi-parallel matching algorithm for the Clos network.

Similar to the *m*-matching algorithm, the Euler partition algorithm uses the edge coloring bipartite graph approach to resolve the routing problem in the Clos network. However, the Euler partition algorithm uses looping to partition the bipartite graph to achieve a fast edge coloring process. First, an example of the Euler partition algorithm. Referring to the bipartite graph in Figure 12.3, the Euler partition algorithm first uses two colors to divide the bipartite graph into two (Fig. 12.7). With reference to Figure 12.8, after Euler partition, one bipartite graph can be divided into two for further assignment. Note that the two partitioned bipartite graphs are independent of each other; thus the second iteration of the Euler partition algorithm can be done in a parallel manner. One thing to notice is that

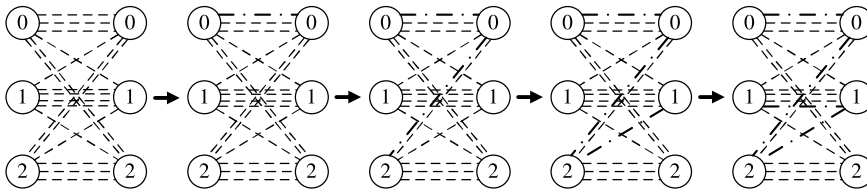


Figure 12.7 Euler partition algorithm, step 1.

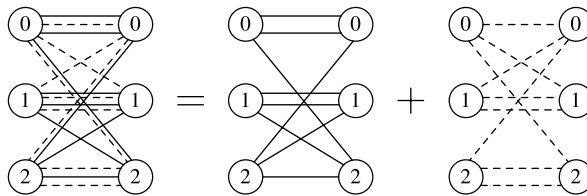


Figure 12.8 Euler partition algorithm, step 2.

the Euler partition algorithm is only efficient for an even degree bipartite graph. In the case of odd degree edges appearing in the bipartite graph, perfect matching to reduce the degree by 1 is used.

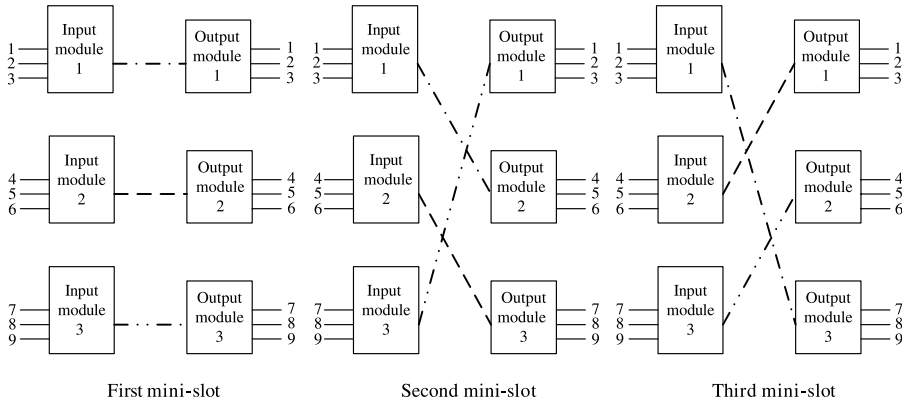
In summary, the Euler partition algorithm is a multi-iteration looping algorithm. Each iteration of the Euler partition algorithm consists of (1) in the case of an even degree bipartite graph, performing a Euler split to partition the graph into two; (2) in the case of an odd degree bipartite graph, performing perfect matching to reduce the degree by 1 and then performing Euler partition. As a result, the total complexity of the Euler partition algorithm is  $O(N \log(m))$ .

The Euler partition algorithm achieves the balance between performance and complexity. However, the Euler partition algorithm's complexity linearly increases with the switch size  $N$ . With an increase of link speed and switch size, it is neither scalable nor practical in today's high-speed networks. From here on, we will discuss six parallel heuristic algorithms for the Clos network. They might not achieve the perfect matching from timeslot to timeslot but they are ultrascaleable and practical to implement in modern high-speed switches.

### 12.5 KAROL'S ALGORITHM

Karol's matching algorithm can assign central routes for cells in a heuristic manner and yet still achieve a good assignment result. Referring to Figure 12.1, let  $i$  be the index of input modules (IMs) where  $i \in 0, 1, 2, 3, \dots, k - 1$ . Let  $j$  be the index of output modules (OMs) where  $j \in 0, 1, 2, 3, \dots, k - 1$  ( $k$  is the number of center modules in the Clos network). In Karol's matching algorithm, each timeslot is divided into  $k$  mini-slots. For  $0 \leq t \leq k - 1$ , in mini-slot  $t$ , IM  $i$  communicate only with OM  $j$  where  $j = [(t + i) \text{ modulo } k]$ . The purpose of this mini-slot matching is to attempt to find central-module routes for those cells from IM  $i$  to OM  $j$ . After  $k$  mini-slots, each IM-OM pair is matched up once and the entire central-module assignment is done in a distributed manner. Because of the unique switch architecture of the Clos-network switch, within each mini-slot central-module routes are independent among



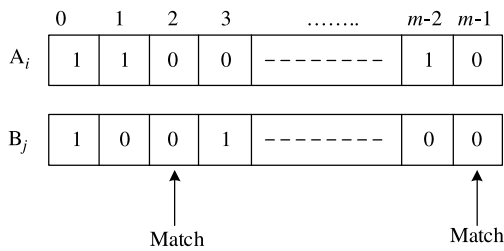


**Figure 12.9** Karol's matching algorithm in 3-stage Clos networks.

all IM–OM pairs, which enables each IM–OM pair to assign central-module routes freely without causing internal blocking. In order to achieve fairness among all traffic loads, the matching sequence of Karol's matching algorithm can be modified and done in a round robin fashion in such a way that for each set of traffic requests, the order of modules matching can be shifted among all matching pairs. For example, suppose that in the current timeslot, IM  $i$  starts its mini-slot matching procedure from OM  $j$ . Then in the next timeslot, it may start from OM  $[(j + 1) \text{ modulo } k]$ , and so on and so forth. As a result, the complexity of Karol's algorithm is only  $O(k)$ .

An example of Karol's matching algorithm in the Clos network is given as follows. Let us consider a  $9 \times 9$  Clos network switch, which has three IMs, three CMs, and three OMs. Therefore, it takes three mini-slots for all IMs to finish performing Karol's matching with all OMs. Figure 12.9 shows Karol's matching algorithm for these three mini-slots.

Further more, Karol's algorithm uses a vector for each input module and each output module to record the availability of the central modules. With reference to Figure 12.10, those vectors are matched up in pairs, each of which has one vector for an input module (e.g.,  $A_i$ ) and the other for an output module (e.g.,  $B_j$ ). Each IM/OM vector is composed of  $m$  bits, and each bit corresponds to a CM. A "0" means that the link to the CM is available and "1" represents unavailable. For those pairs of modules that have a cell to dispatch between them, the two vectors will be compared to locate a "0," that is, an available link to a CM, if any.



**Figure 12.10** Vector representation of center route availability in Karol's algorithm.  $A_i$  is a  $m$ -bit vector of  $IM_i$ ; each bit indicates the availability of the link to the CM.  $B_j$  is also a  $m$ -bit vector of  $OM_j$ ; each bit indicates the availability of the link from the CM.

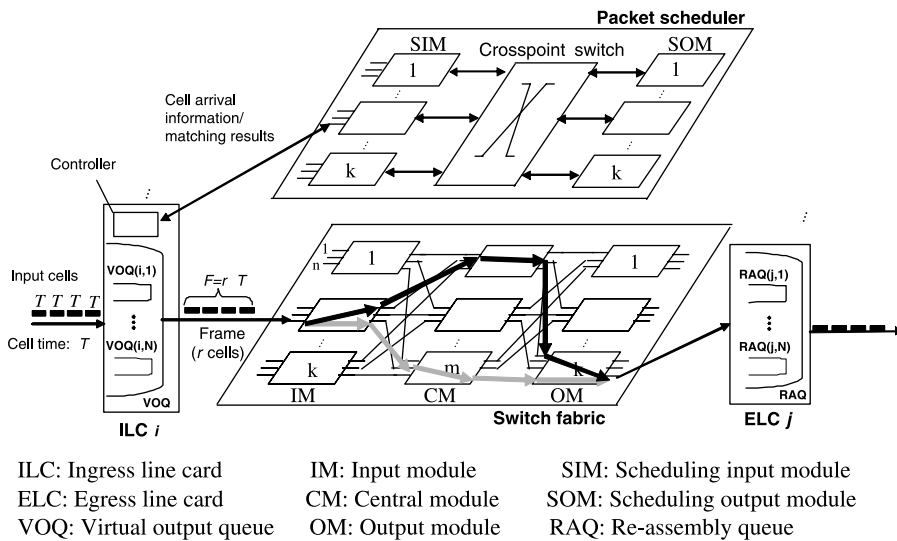
In the next three sections, a new class of matching algorithms, called MAC, for resolving scheduling problems in 3-stage Clos-network switches are presented. To relax the strict arbitration time constraint, MAC operates based on a frame of  $r$  cells ( $r > 1$ ).

### 12.6 FRAME-BASED MATCHING ALGORITHM FOR CLOS NETWORK (f-MAC)

Figure 12.11 shows the structure of a MAC packet switch, which consists of a packet scheduler (PS) and a 3-stage Clos-network switch fabric. The PS consists of  $k$  scheduling input modules (SIMs) and  $k$  scheduling output modules (SOMs), each of which corresponds to an input switch module (or output switch module) in the 3-stage Clos-network switch. A crosspoint switch with a reconfigured pattern is used to interconnect these SIMs and SOMs.

All incoming packets are first terminated at ingress line cards (ILCs), where they are segmented into cells (fixed length data units) and stored in a memory. The packet headers are extracted for IP address lookup, classification, and other functions such as traffic shaping/policing. Packets destined for the same output port are stored in the same virtual output queues (VOQs) in the ILCs. Multiple cells (e.g.,  $r$  cells) in the same VOQ form a frame that is sent to an output port once a grant signal is given by the PS. Let us define a cell time slot to be  $T$ , and the frame period  $F = r \times T$ , where  $r$  is the frame size in number of cells. The ILCs send cell-arrival information to the PS. The PS then resolves contention, finds routing paths in the center stage, and sends grant signals to the ILCs in each extended frame period  $F$ . A large buffer with a re-assembly queue (RAQ) structure is used in the egress line card (ELC) to store the frames from the switch fabric and to re-assemble them into packets.

The first MAC algorithm to be discussed is the frame-based matching algorithm for Clos-network switch (f-MAC). The f-MAC includes two phases to solve the switching problems. It first resolves the contention of the frames from different input ports that are



**Figure 12.11** Structure of a 3-stage Clos-network switch and a packet scheduler.

destined for the same output port. It then determines a routing path through the center stage (i.e., chooses a CM) for each matched input–output pair. Since there can be multiple possible paths (determined by the number of CMs) for each matched I/O, choosing a CM to reduce internal blocking and thus increase the throughput further complicates the scheduling algorithm design.

In the first phase, f-MAC is an extension of the exhaustive dual round-robin matching (EDRRM) scheme [6], by including the frame concept. Most iterative matching schemes, such as *i*SLIP [7] and DRRM [3], suffer from the problem of throughput degradation under unbalanced traffic distribution. The EDRRM scheme improves throughput by maintaining the existing matched pairs between the inputs and outputs so that the number of unmatched inputs and outputs is drastically reduced (especially at high load), thus reducing the inefficiency caused by not being able to find matches among those unmatched inputs and outputs. The f-MAC also modifies EDRRM slightly to further improve the throughput. One of the major problems of the exhaustive matching is that it may cause starvation in some inputs. One way to overcome this problem is to set a timer for each head-of-line (HOL) frame. When the timer expires, the request from the “expired” frame is given the highest preference.

In phase 1, f-MAC finds I/O matching and consists of three steps:

**Step 1: *Request*.** Each unmatched input sends a request to every output port arbiter for which it has a queued cell in the corresponding VOQ. The request is set at high priority if the queue length is above (or equal to) the threshold  $r$ ; otherwise, the request is set at low priority. Each matched input only sends a high-priority request to its matched output.

**Step 2: *Grant*.** If each output port arbiter receives one or more high-priority requests, it chooses the first request to grant starting from the current position of the high-priority pointer. Otherwise, it grants the first low-priority request starting from the current position of the low-priority pointer.

**Step 3: *Accept*.** If each input port arbiter receives one or more high-priority grants, it accepts the first starting from the current position of the high-priority pointer. Otherwise, it accepts the first starting from the current position of the low-priority pointer.

The pointers of the input port arbiter and output port arbiter are updated to the next position only if the grant is accepted in step 3.

After input–output matching is completed in phase 1, f-MAC finds a routing path in phase 2 for each matched input–output pair through the 3-stage bufferless Clos-network switch. To reduce computation complexity, a simple parallel matching scheme [9] is adopted. That is, f-MAC includes  $k$  matching cycles. In each matching cycle, each SIM is matched with one of the  $k$  SOMs and the parallel matching scheme described in Section 12.5 is adopted to find the vertical pairs of zeros between  $A_i$  and  $B_j$ .

## 12.7 CONCURRENT MATCHING ALGORITHM FOR CLOS NETWORK (c-MAC)

With the increase of switch sizes and port speeds, the hardware and interconnection complexity between input and output arbiters makes it very difficult to design the packet scheduler in a centralized way. This section presents a more scalable concurrent matching

algorithm for Clos-network switches, called c-MAC. It is highly distributed such that the input–output matching and routing-path finding are concurrently performed by scheduling modules.

Figure 12.12 shows the architecture of the packet scheduler. It consists of  $k$  SIMs and  $k$  SOMs, each of which corresponds to an input switch module (or output switch module) in the 3-stage Clos-network switch (see Fig. 12.11). There are  $n$  IPAs in each SIM. Each SIM consists of  $n$  virtual output port arbiters (VOPAs), each of which corresponds to an output port in the corresponding OM. Each SIM has an input module arbiter (IMA), and each SOM has an output module arbiter (OMA). A crosspoint switch with a predetermined reconfigured pattern is used to interconnect these SIMs and SOMs. As shown in Figure 12.11, each ILC has  $N$  VOQs, each corresponding to an ELC. A counter  $C(i, j)$  in the PS is used to record the number of cells in the corresponding  $VOQ(i, j)$ .

The c-MAC scheme divides one frame period  $f$  into  $k$  matching cycles as shown in Figure 12.13. In each matching cycle, each SIM is matched with one of the  $k$  SOMs. During each cycle, c-MAC includes two phases to find the input–output matches and routing paths, respectively.

At the beginning of each matching cycle, each SOM passes  $m + 2n$  bits to the corresponding SIM, where the  $m$  bits correspond to the state of  $m$  input links of the corresponding OM; the  $2n$  bits correspond to the state of  $n$  output ports of the corresponding OM. There are four possible states for each output port: “00” when the output port is unmatched; “01” when the output port is matched with low priority in the last frame period; “10” when the output port is matched with high priority in the last frame period; “11” when the output port is matched in this frame period.

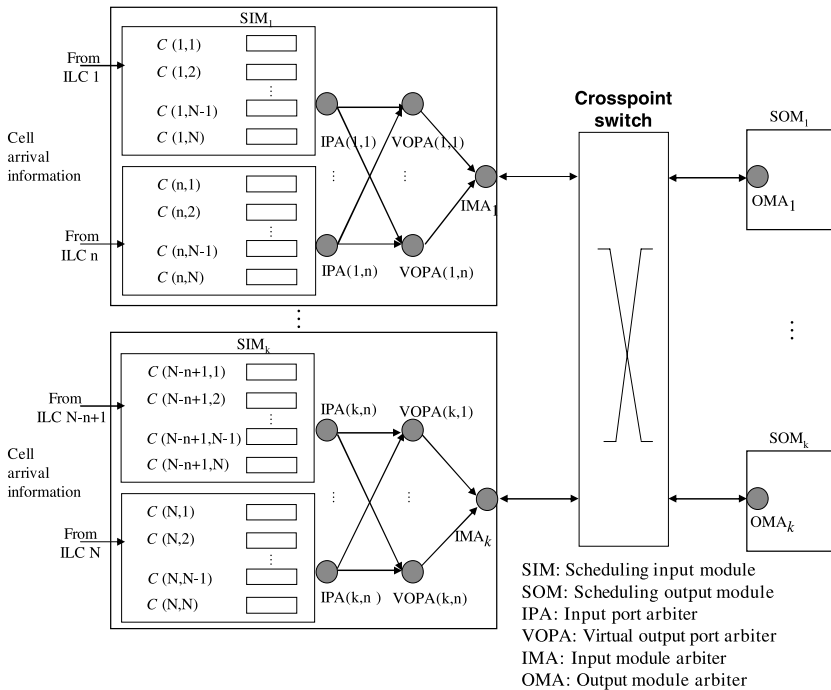


Figure 12.12 Schematic of the packet scheduler.

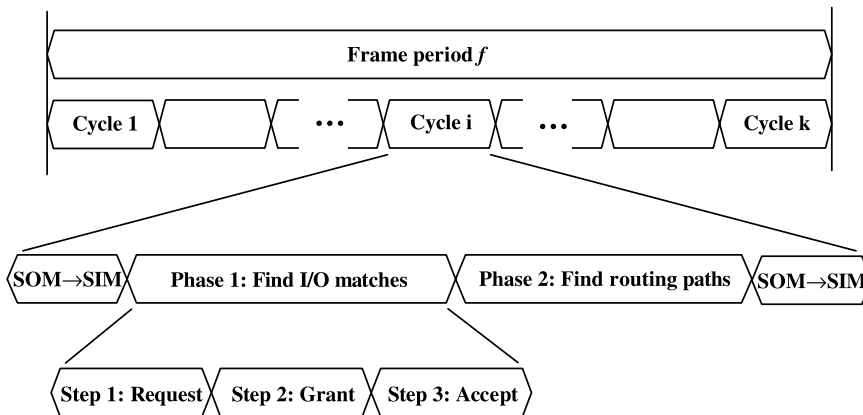


Figure 12.13 Timing schematic of the c-MAC scheme.

It is assumed that the matching sequence between SIMs and SOMs is predetermined. For instance, in the first cycle,  $SIM_i$  is matched with  $SOM_j$ , where  $1 \leq i \leq k$  and  $1 \leq j \leq k$ . In the second cycle,  $SIM_i$  is matched with  $SOM_{(j \bmod k)+1}$ . The procedure is repeated  $k$  times. To achieve matching uniformity for all the SIMs, the beginning matching sequence between SIMs and SOMs is skewed one position at the beginning of each frame period.

Phase 1 finds I/O matching and consists of three steps as described below:

**Step 1: Request.** Each matched input port arbiter (IPA) only sends a high-priority request to its matched VOPA; each unmatched IPA (including the currently matched IPA but whose matched VOQ's queue length is less than a threshold  $r$ ) sends a 2-bit request to every VOPA for which it has queued cells in the corresponding VOQ. ("00" means no request; "01" means low-priority request because queue length is less than  $r$ ; "10" means high-priority request because queue length is larger than  $r$ ; "11" means the highest priority because the waiting time of the HOL frame is larger than a threshold,  $T_w$ .) Note that, using the waiting time mechanism for the HOL frames prevents the starvation problem.

**Step 2: Grant.** Only the "available" VOPA performs the grant operation. A VOPA is defined to be "available," if its corresponding output port is

- (a) Unmatched; or
- (b) Matched in the last frame period with low priority (the VOPA receives at least one high-priority request at this frame period); or
- (c) VOPA is matched in the last frame period with high priority, but it receives the request from the matched IPA and its priority is becoming low-priority in this frame period.

If a VOPA is "available" and receives one or more high-priority requests, it grants the one that appears next in a fixed round-robin schedule starting from the current position of the high-priority pointer. If there are no high-priority requests, the output port arbiter grants one low-priority request in a fixed round-robin schedule starting from the current position of the low-priority pointer. The VOPA notifies each requesting IPA whether or not its request is granted.

Step 3: *Accept*. If the IPA receives one or more high-priority grants, it accepts the one that appears next in a fixed round-robin schedule starting from the current position of the high-priority pointer. If there are no high-priority grants, the input port arbiter accepts one low-priority request in a fixed round-robin schedule starting from the current position of the low-priority pointer. The input port arbiter notifies each VOPA whether or not its grant is accepted. Update of the pointers: The pointer of IPA and VOPA is updated to the chosen position only if the grant is accepted in Step 3 of phase 1 and also accepted in phase 2.

In phase 2, c-MAC adopts the parallel matching scheme [9] to find the routing paths for the matched I/O pairs as described in previous section.

## 12.8 DUAL-LEVEL MATCHING ALGORITHM FOR CLOS NETWORK (d-MAC)

In c-MAC, each pair of SIM–SOM needs to be matched once, regardless of the queuing status of the switch, yielding  $k$  matching cycles for the arbitration in one frame period. This, however, results in a high time complexity that prevents the selection of a small frame size. To further reduce the scheduling time complexity and relax the arbitration time constraint, we have proposed a new dual-level matching algorithm for the Clos-network switch, call d-MAC, to determine the matching sequence between the SIMs and SOMs according to the queuing information of the switch.

The d-MAC scheme consists of two levels of matching, that is, module-level matching and port-level matching. The former is responsible for determining the SIM–SOM matching pattern according to the queuing status of the switch. The latter is responsible for determining the port-to-port matching and finding the internal routing path for the matched input–output pair, as the task of matching cycle in c-MAC.

For the module-level matching, the switching patterns of a number of, say  $F$ , frames can be determined simultaneously. These  $F$  frames constitute a super-frame. We use the example shown in Figure 12.14 to illustrate the module-level matching steps of d-MAC as follows. With reference to Figure 12.14a, a traffic matrix is used to represent the queuing status of the switch, in which entry  $(i, j)$  denotes the number of buffered cells that desire to be transmitted from  $IM_i$  to  $OM_j$ . According to this traffic matrix, a request matrix can be obtained. Note that the request matrix gives the number of requests that can be sent from each SIM to each SOM. Then a scheduling algorithm can be employed to do arbitration among these requests, and this process produces a super-frame matrix, in which each entry represents the matching opportunities between each SIM and each SOM in one super-frame. With reference to Figure 12.14b, the super-frame matrix is further decomposed into the module-level matching matrices, where a module-level matching matrix represents the matching pattern of SIMs and SOMs in one matching cycle as shown in Figure 12.14c. The module-level matching is done after the module-level matching matrices are determined. With each of these matrices, the port-level matching can thereafter perform the task of matching cycle, that is, port-to-port matching and route assignment, for the given SIM–SOM pairs.

The module-level matching and port-level matching assignment can be performed in a pipelined manner as shown in Figure 12.15. As described above, each super-frame is composed of  $F$  frames, and each frame consists of  $r$  cells. Suppose that the number of matching cycles in each frame is set to be  $k'$ , where  $k' \leq k$ , the transmission time of  $r$

$$\begin{pmatrix} 16 & 1 & 11 & 2 \\ 2 & 12 & 0 & 4 \\ 7 & 10 & 11 & 5 \\ 2 & 6 & 7 & 8 \end{pmatrix} \Rightarrow \begin{pmatrix} 4 & 1 & 3 & 1 \\ 1 & 3 & 0 & 2 \\ 2 & 3 & 3 & 2 \\ 1 & 2 & 2 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 1 & 2 & 1 \\ 1 & 3 & 0 & 2 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

Traffic matrix                      Request matrix                      Super-frame matrix

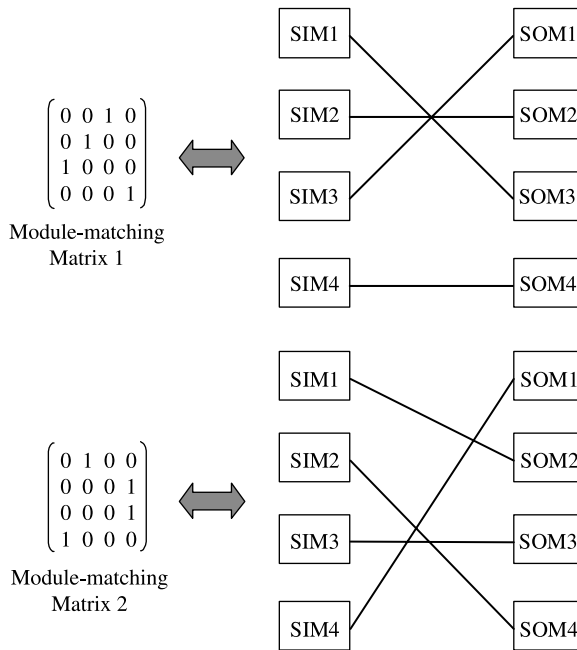
(a) Determining super-frame matrix

$k'$  matching cycles in one frame

$$\begin{pmatrix} 2 & 1 & 2 & 1 \\ 1 & 3 & 0 & 2 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} + \dots + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Super-frame matrix                      Module-matching matrices

(b) Super-frame matrix decomposition



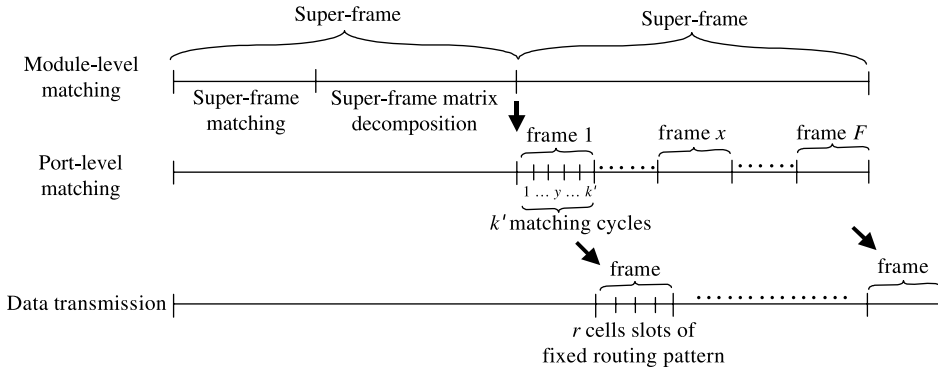
(c) Correspondence between Module-matching matrix and Matching pattern of SIMs/SOMs

**Figure 12.14** Illustration of module-level matching of d-MAC ( $k = 4, k' = 2, F = 3$ ).

cells must be greater than or equal to the arbitration time of  $k'$  matching cycles. This is the constraint against selecting a small  $r$ . In each super-frame, there are a total of  $F \times k'$  matching cycles. Thus  $F$  sets of  $k'$  module-matching matrices are to be determined for the next super-frame, while the scheduler is doing the port-level matching for the current super-frame.

The module-level matching algorithm is composed of two phases, that is, super-frame matching and super-frame decomposition.

The super-frame matching is to determine a super-frame matrix in accordance with the queuing status between the switch modules. Each entry in the super-frame matrix represents



**Figure 12.15** Pipelined process of the d-MAC matching scheme.

the matching opportunities between each SIM and each SOM in one super-frame. A super-frame matrix is a  $k \times k$  matrix with (i) no row/column sum greater than  $F \times k'$ , and (ii) no entry greater than  $F$ . To determine the super-frame matrix, the d-MAC scheme adopts an iterative request/grant/accept algorithm, modified from the *i*SLIP scheme.

The super-frame decomposition is to decompose the super-frame matrix into  $F \times k'$  module-matching matrices. Each module-matching matrix records the matching status between the SIMs and SOMs in one matching cycle of the next super-frame. The matrix-decomposition problems can be solved by the edge-coloring algorithms. However, the optimal edge-coloring algorithms are not preferable here because of their high time complexity. Instead, we use a parallel matching heuristic [9] to decompose the super-frame matrix. This algorithm contains  $k$  rounds. In each round, each SIM is communicating with one of  $k$  SOMs. Each SIM/SOM maintains a two-tuple array that contains  $F \times k'$  zero-one variables. Let  $W_i(Z_j)$  be the array of  $SIM_i(SOM_j)$

$$W_i(x, y) = \begin{cases} 0, & \text{if } SIM_i \text{ is unmatched in cycle } y \text{ of frame } x, \\ 1, & \text{if it has been matched in cycle } y \text{ of frame } x; \end{cases}$$

$$Z_j(x, y) = \begin{cases} 0, & \text{if } SIM_j \text{ is unmatched in cycle } y \text{ of frame } x, \\ 1, & \text{if it has been matched in cycle } y \text{ of frame } x; \end{cases}$$

where  $1 \leq x \leq F$  and  $1 \leq y \leq k'$ .

When  $SIM_i$  is communicating with  $SOM_j$ , the d-MAC scheme tries to find as many common zero entries in  $W_i$  and  $Z_j$  as possible to meet the number given by entry  $(i, j)$  in the super-frame matrix. Note that no more than one “0” entry in the same frame can be assigned to the same SIM–SOM pair.

When the module-level matching is completed, the matching sequence between SIMs and SOMs (recorded in the module-matching matrices) is determined for the next super-frame. The port-level matching algorithm consists of  $k'$  matching cycles in a frame. In each matching cycle, the port-level matching algorithm includes two steps: (i) the port-to-port matching assignment; (ii) the central module assignment.

To find the port-to-port matching for the corresponding pair of IM–OM, the d-MAC scheme adopts an iterative request/grant/accept algorithm, for example, *i*SLIP. To improve the matching efficiency, the d-MAC scheme also introduces high-priority and low-priority



arbiters in the SIMs. Under the priority mechanism, the VOQs with queue lengths of more than  $r$  cells will send out high-priority requests and have higher priority than the unfilled ones. To determine an internal routing path for each matched input–output pair, the d-MAC scheme adopts a heuristic parallel matching algorithm [6] described in Section 12.5.

In the above sections, we discussed various scheduling algorithms for bufferless Clos network switches in which all scheduling is done upon the packet entering the input modules of the switch. In the next two sections, we present other two scheduling algorithms for buffered Clos network switch in which the added buffer in the Clos network switch relaxes the scheduling so that the architecture and scheduling are more feasible for practical implementation.

## 12.9 THE ATLANTA SWITCH

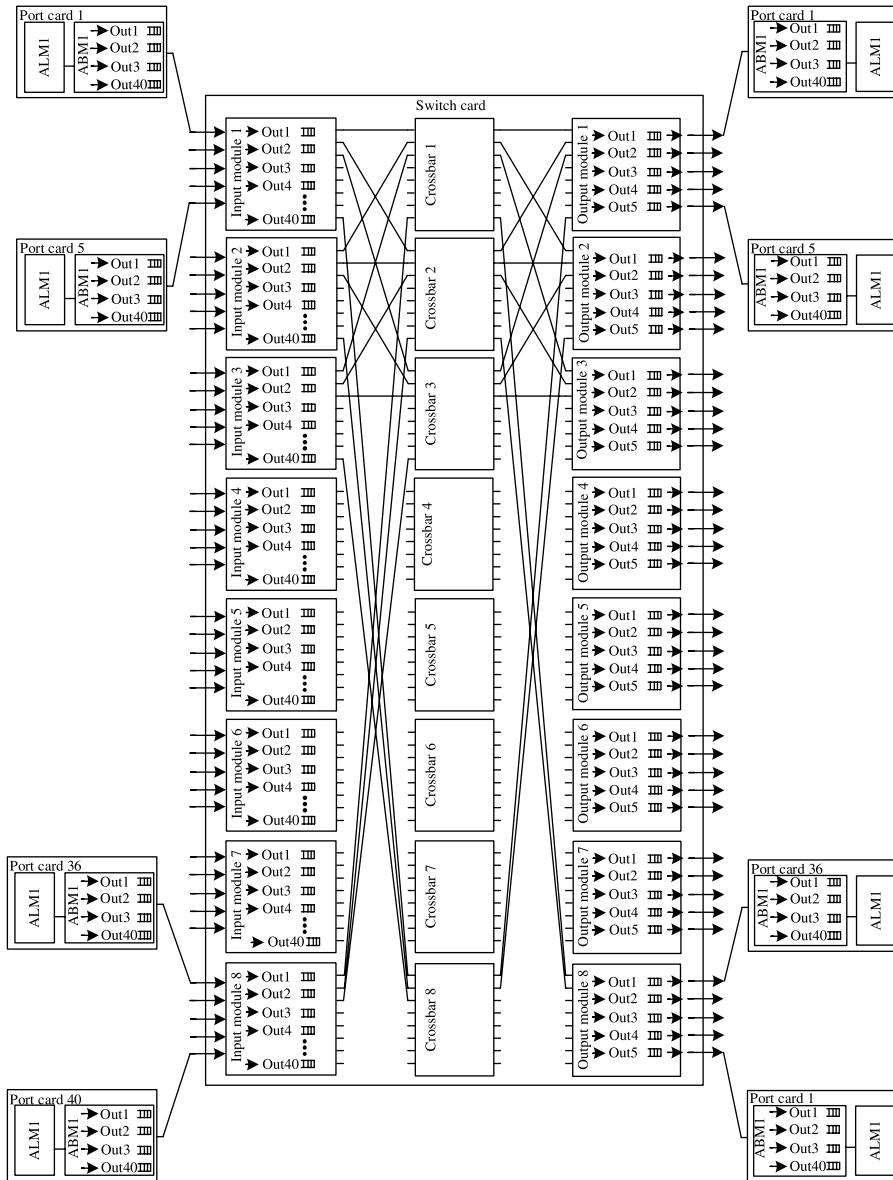
The ATLANTA switch architecture has a three-stage multi-module memory/space/memory (MSM) arrangement. The MSM configuration uses buffers in the input and output stages, while the second stage is bufferless. A simple distributed self-routing algorithm is used to dispatch cells from the input to the output stage. Although cells are routed individually and multiple paths are provided from each input to each output, cell sequence is preserved due to its bufferless second stage. Selective backpressure is used from the output to the input buffers in the fabric, so the required buffers in the output stage are also relatively small.

The ATLANTA architecture provides support for multicast traffic. Cells belonging to a multicast virtual circuit are always replicated according to a minimum multicast tree, that is, they are replicated as far downstream as possible in the switch; this minimizes the amount of resources required to sustain the expansion in traffic volume internally to the switch due to multicasting. A single copy of a multicast cell is locally stored in each buffer, and replicated (if necessary) only when the cell is sent to the following stage in the switch or to its desired destinations.

In the following, we describe the operation of the MSM switch fabric with reference to the specific  $40 \times 40$  configuration shown in Figure 12.16. The MSM configuration is based on three main principles:

- By using buffers in the first stage to store cells that cannot be routed to the output buffers at a given time, the number of paths necessary for nonblocking behavior can be greatly reduced.
- By using a bufferless center stage, cells belonging to the same virtual circuit can be routed individually without affecting cell sequence.
- By using selective backpressure from the output buffers to the input buffers, buffers can be located where they are most economical.

Under these design principles in the ATLANTA switch, the memory switch is used to implement the switching modules in the first and third stages, while crossbars are implemented in the second stage. Each module in the first and third stages must be connected to all crossbars. All interconnection lines between adjacent stages have the same rate as the input and output ports. To realize nonblocking in the MSM configuration, it is well known that its internal capacity must be higher than the aggregate capacity of the input ports. We call this “expansion.” The required expansion is achieved by connecting fewer than eight ports to each input and output module. In the  $40 \times 40$  configuration of Figure 12.16, five



**Figure 12.16** Schematic configuration of a  $40 \times 40$  multistage ATLANTA switch. (©1997 IEEE.)

ports are connected to each edge module for an expansion factor equal to  $5 : 8$ . The expansion ratio is  $1.6 (=8/5)$ . Each module in the first stage maintains 40 groups of queues; each group corresponds to one of the output ports in the switch. Each module in the third stage manages a number of groups equal to the number of ports connected to that module (in this case five).

In order to minimize the required expansion, an efficient routing algorithm is necessary to route cells from the input to the output modules. Intuitively, the idea is that the fabric is nonblocking as long as the equivalent service capacity (i.e., the maximum switching

capacity provided by the expansion and the routing algorithm) in the input queues is higher than the aggregate input capacity of those queues. A practical constraint for such a system to be cost-effective is that the routing algorithm must be fully distributed and independently run by each input module. The concurrent dispatching algorithm that is used in the ATLANTA architecture is now discussed.

The concurrent dispatching works as follows. In each time slot, each input module in the first stage selects up to eight cells to be served in that time slot. The selection process over the 40 groups of queues uses a two-level weighted-round-robin mechanism.<sup>1</sup> Once the cells are selected, each input module sends up to eight bids to the crossbars, one per each crossbar. A bid contains the desired destination and service priority of one of the selected cells. Since there is no coordination among the input modules, a crossbar can receive more than one bid for the same output module at a time. In case of conflict between two or more bids, the crossbar selects one as the winning bid. In selecting the winning bid, and generally in determining whether a bid is successful or not, the crossbar takes into account whether or not the specific queue in the output module requested by each bid has available buffer space (the third-stage modules continuously send backpressure information to the crossbars informing them of the availability of buffers for each queue), and never declares successful a bid that requests a queue with no available buffer space. Then the crossbar sends a feedback signal to the input modules, informing each module whether or not the bid was successful, and in the latter case whether the bid was unsuccessful because of lost contention in the crossbar or due to selective backpressure from the output modules.

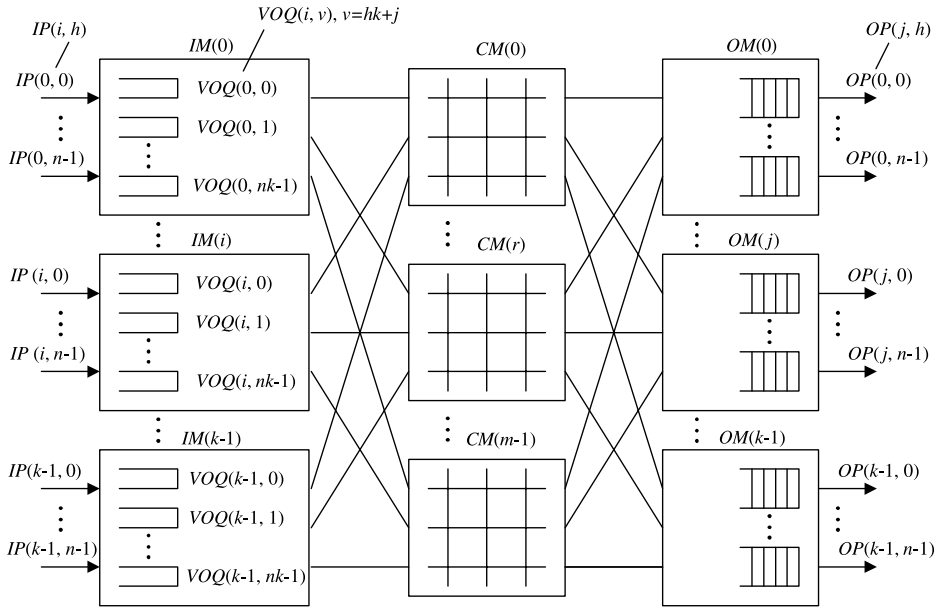
If the bid was successful, in the following time slot the input module transmits the corresponding cell through the crossbar; in the same time slot, the input module also selects another cell and initiates a new bidding process for it on that crossbar. If the bid was unsuccessful because of lost contention in the crossbar, the input module again sends that same bid to the crossbar in the following time slot. If the bid was unsuccessful because of backpressure from the output modules, the input module selects a different cell from the buffer and initiates a bidding process for it in the following time slot.

## 12.10 CONCURRENT ROUND-ROBIN DISPATCHING (CRRD) SCHEME

To achieve 100 percent throughput by using the random dispatching scheme, the internal expansion ratio is set to about 1.6 when the switch size is large [8]. Here, we describe a concurrent round-robin dispatching (CRRD) scheme [5] that can achieve 100 percent throughput under uniform traffic. The basic idea of CRRD is to use the desynchronization effect in the Clos-network switch. The desynchronization effect has been studied using simple scheduling algorithms as *i*SLIP [7] and dual round-robin matching (DRRM) [6] in an input-queued crossbar switch. CRRD provides high switch throughput without expanding the internal bandwidth, while the implementation is simple because only simple round-robin arbiters are employed.

**Basic Architecture.** Figure 12.17 shows the CRRD switch. The terminology used in this section is as follows:

<sup>1</sup>Cells in each group are further classified into several categories of different service priority.



**Figure 12.17** CRRD switch with virtual output queues (VOQs) in the input modules.

- IM** Input module at the first stage.
- CM** Central module at the second stage.
- OM** Output module at the third stage.
- n* Number of input ports/output ports in each IM/OM, respectively.
- k* Number of IMs/OMs.
- m* Number of CMs.
- i* IM number, where  $0 \leq i \leq k - 1$ .
- j* OM number, where  $0 \leq j \leq k - 1$ .
- h* Input port (IP)/output port (OP) number in each IM/OM, respectively, where  $0 \leq h \leq n - 1$ .
- r* Central-module (CM) number, where  $0 \leq r \leq m - 1$ .
- IM(i)* *i*th IM.
- CM(r)* *r*th CM.
- OM(j)* *j*th OM.
- IP(i, h)* *h*th input port at *IM(i)*.
- OP(j, h)* *h*th output port at *OM(j)*.
- VOQ(i, v)* VOQ at *IM(i)* that stores cells destined for *OP(j, h)*, where  $v = hk + j$  and  $0 \leq v \leq nk - 1$ .
- G(i, j)* VOQ group at *IM(i)* that consists of *n* *VOQ(i, j, h)*s.
- L<sub>i</sub>(i, r)* Output link at *IM(i)* that is connected to *CM(r)*.
- L<sub>c</sub>(r, j)* Output link at *CM(r)* that is connected to *OM(j)*.

The first stage consists of *k* IMs, each of which has an  $n \times m$  dimension. The second stage consists of *m* bufferless CMs, each of which has a  $k \times k$  dimension. The third stage consists of *k* OMs, each of which has an  $m \times n$  dimension.

An  $IM(i)$  has  $nk$  virtual output queues (VOQs) to eliminate HOL blocking. A VOQ is denoted as  $VOQ(i, v)$ . Each  $VOQ(i, v)$  stores cells that go from  $IM(i)$  to the output port  $OP(j, h)$  at  $OM(j)$ , where  $v = hk + j$ . A VOQ can receive, at most,  $n$  cells from  $n$  input ports in each cell time slot. The HOL cell in each VOQ can be selected for transmission across the switch through  $CM(r)$  in each time slot. This ensures that cells are transmitted from the same VOQ in sequence.

Each  $IM(i)$  has  $m$  output links. An output link  $L_i(i, r)$ , is connected to each  $CM(r)$ . A  $CM(r)$  has  $k$  output links, each of which is denoted as  $L_c(r, j)$ , and it is connected to  $k$  OMs, each of which is  $OM(j)$ . An  $OM(j)$  has  $n$  output ports, each of which is  $OP(j, h)$  and has an output buffer. Each output buffer receives at most  $m$  cells at one time slot, and each output port at OM forwards one cell in a first-in-first-out (FIFO) manner to the output line.

**CRRD Algorithm.** Figure 12.18 illustrates the detailed CRRD algorithm. To determine the matching between a request from  $VOQ(i, v)$  and the output link  $L_i(i, r)$ , CRRD adopts an iterative matching within  $IM(i)$ . An IM has  $m$  output link arbiters, each of which is associated with each output link, and each VOQ has a VOQ arbiter as shown in Figure 12.18.

Two phases of dispatching cells from the first stage to the second stage are considered. In phase 1, at most  $m$  VOQs are selected as candidates and the selected VOQ is assigned to an IM output link. A request that is associated with this output link is sent from IM to CM. This matching between VOQs and output links is performed only within IM. In phase 2, each selected VOQ that is associated with each IM output link sends a request from IM to CM. CMs respond with the arbitration results to IMs so that the matching between IMs and CMs can be done.

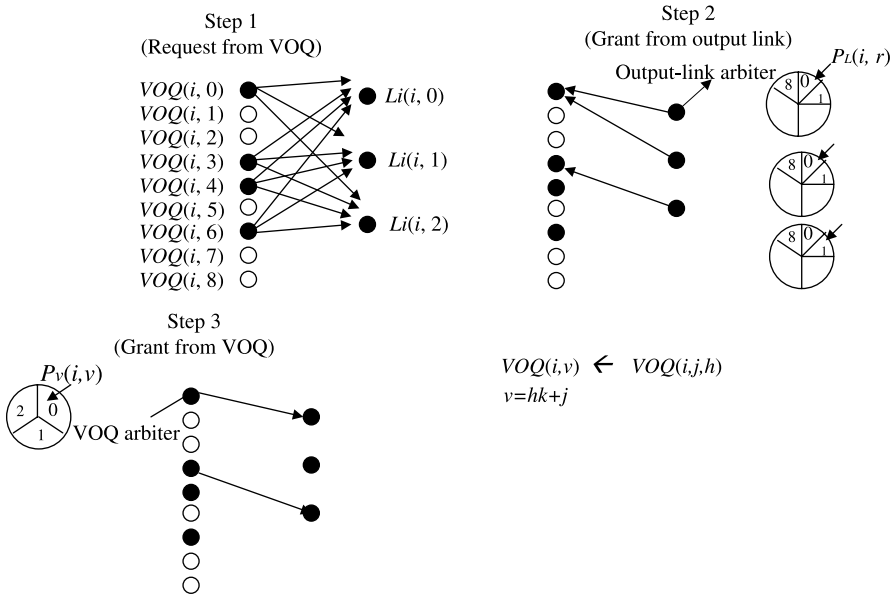


Figure 12.18 CRRD scheme.

### Phase 1. Matching within IM

#### First iteration

Step 1: Each non-empty VOQ sends a request to every output-link arbiter, each of which is associated with  $L_i(i, r)$ , where  $0 \leq i \leq k - 1$  and  $0 \leq r \leq m - 1$ .

Step 2: Each output link  $L_i(i, r)$ , where  $0 \leq i \leq k - 1$  and  $0 \leq r \leq m - 1$ , independently searches a request among  $nk$  non-empty VOQs. Each output-link arbiter associated with  $L_i(i, r)$  has its own pointer  $P_L(i, r)$ , where  $0 \leq i \leq k - 1$  and  $0 \leq r \leq m - 1$ . The output-link arbiter starts to search one non-empty VOQ request from the  $P_L(i, r)$  in a round-robin fashion. Each output-link arbiter sends the grant to a requesting VOQ. Each VOQ has its own round-robin arbiter, and one pointer  $P_v(i, v)$ , where  $0 \leq v \leq nk - 1$  to choose one output link. The VOQ arbiter starts to search one grant out of several grants that are given by the output-link arbiters from the position of  $P_v(i, v)$ .

Step 3: The VOQ that chooses one output link  $L_i(i, r)$  by using the round-robin arbiter sends the grant to the selected output link. Note that the pointer  $P_L(i, r)$  that is associated with each output link and  $P_v(i, v)$  that is associated with each VOQ are updated to one position after the granted position, only if they are matched and the request is also granted by CM in phase 2.

#### $i$ th iteration ( $i > 1$ )

Step 1: Each unmatched VOQ at the previous iterations sends a request to all the output-link arbiters again.

Steps 2 and 3: Follow the same procedure as in the first iteration.

### Phase 2. Matching between IM and CM


Step 1: After phase 1 is completed, output link  $L_i(i, r)$  sends the request to the CM. Then contention control in the CM is performed. Each  $CM(r)$  has  $k$  pointers  $P_c(r, j)$ , where  $0 \leq r \leq m - 1$  and  $0 \leq j \leq k - 1$ , each of which corresponds to each  $OM(j)$ . The CM makes its arbitration using the pointer  $P_c(r, j)$  in a round-robin fashion, and sends the grants to  $L_i(i, r)$  of  $IM(i)$ . The pointer  $P_c(r, j)$  is updated when the CM sends the grant to the IM.

Step 2: If the IM receives the grant from the CM, it sends a corresponding cell from that VOQ at the next time slot. Otherwise, the IM will not send a cell at the next time slot. The request that is not granted from the CM will be dispatched again at the next time slot because the pointers that are related to the ungranted requests are not updated.

The CRRD algorithm has to be completed within one time slot to provide the matching result every time slot.

Figure 12.18 shows an example of  $n = m = k = 3$ , where CRRD is operated at the first iteration in phase 1. At step 1,  $VOQ(i, 0)$ ,  $VOQ(i, 3)$ ,  $VOQ(i, 4)$ , and  $VOQ(i, 6)$ , which are non-empty VOQs, send requests to all the output-link arbiters. At step 2, output-link arbiters associated with  $L_i(i, 0)$ ,  $L_i(i, 1)$ , and  $L_i(i, 2)$ , select  $VOQ(i, 0)$ ,  $VOQ(i, 0)$ , and  $VOQ(i, 3)$ , respectively, according to their pointers' positions. At step 3,  $VOQ(i, 0)$  receives two grants from both output-link arbiters of  $L_i(i, 0)$  and  $L_i(i, 1)$ , selects  $L_i(i, 0)$  by using its own VOQ arbiter, and sends a grant to the output-link arbiter of  $L_i(i, 0)$ . Since  $VOQ(i, 3)$  receives one grant from an output-link arbiter  $L_i(i, 2)$ , it sends a grant to the output-link arbiter. With one

	$T$	0	1	2	3	4	5	6	7
$IM(0)$	$P_V(0,0)$	0	1	0	0	0	1	0	0
	$P_V(0,1)$	0	0	1	0	0	0	1	0
	$P_V(0,2)$	0	0	0	1	0	0	0	1
	$P_V(0,3)$	0	0	0	0	1	1	0	0
$IM(1)$	$P_V(1,0)$	0	0	1	0	0	0	1	0
	$P_V(1,1)$	0	0	0	1	0	0	0	1
	$P_V(1,2)$	0	0	0	0	1	0	0	0
	$P_V(1,3)$	0	0	0	0	0	1	0	0
$IM(0)$	$P_L(0,0)$	0	1	2	3	0	1	2	3
	$P_L(0,1)$	0	0	1	2	3	0	1	2
$IM(1)$	$P_L(1,0)$	0	0	1	2	3	0	1	2
	$P_L(1,1)$	0	0	0	1	2	3	0	1
$CM(0)$	$P_C(0,0)$	0	1	0	1	0	1	0	1
	$P_C(0,1)$	0	0	1	0	1	0	1	0
$CM(1)$	$P_C(1,0)$	0	0	1	0	1	0	1	0
	$P_C(1,1)$	0	0	0	1	0	1	0	1

 The request is granted by CM.

**Figure 12.19** Example of desynchronization effect in CRRD ( $n = m = k = 2$ ).

iteration,  $L_i(i, 1)$  cannot be matched with any non-empty VOQs. At the next iteration, the matching between unmatched non-empty VOQs and  $L_i(i, 1)$  will be performed.

**Desynchronization Effect of CRRD.** While the ATLANTA switch suffers contention at CM [8], CRRD decreases the contention at the CM because pointers  $P_V(i, v)$ ,  $P_L(i, r)$ , and  $P_C(r, j)$ , are desynchronized.

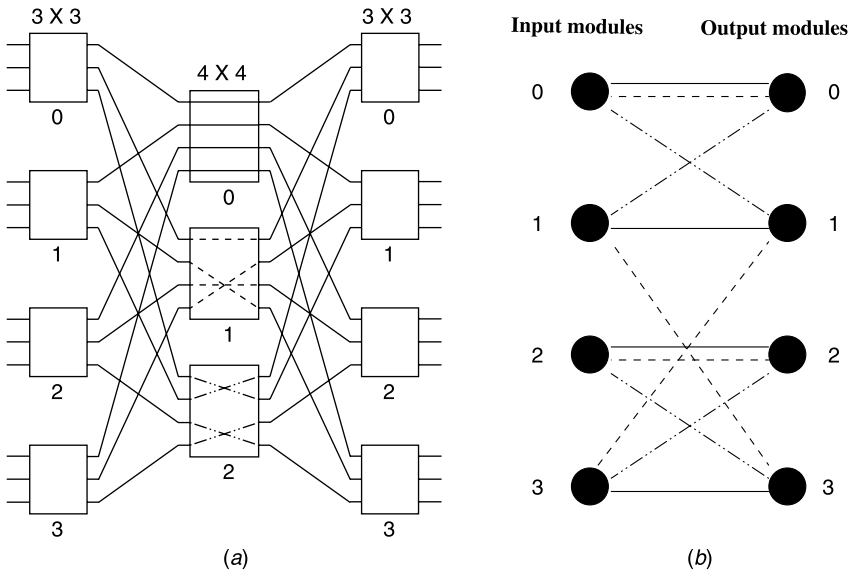
By using a simple example, desynchronization of the pointers is demonstrated. Consider the example of  $n = m = k = 2$  as shown in Figure 12.19. We can assume that every VOQ is always occupied with cells. Each VOQ sends a request to be selected as a candidate at every time slot. All the pointers are set to be  $P_V(i, v) = 0$ ,  $P_L(i, r) = 0$ , and  $P_C(r, j) = 0$  at the initial state. Only one iteration in phase 1 is considered here.

At time slot  $T = 0$ , since all the pointers are set to 0, only one VOQ in  $IM(0)$ , which is  $VOQ(0, 0, 0)$ , can send a cell with  $L_i(0, 0)$  through  $CM(0)$ . The related pointers with the grant,  $P_V(0, 0)$ ,  $P_L(0, 0)$ , and  $P_C(0, 0)$ , are updated from 0 to 1. At  $T = 1$ , three VOQs, which are  $VOQ(0, 0, 0)$ ,  $VOQ(0, 1, 0)$ , and  $VOQ(1, 0, 0)$ , can send cells. The related pointers with the grants are updated. Four VOQs can send cells at  $T = 2$ . In this situation, 100 percent switch throughput is achieved. There is no contention at the CMs from  $T = 2$  because the pointers are desynchronized.

### 12.11 THE PATH SWITCH

If we consider each input module and each output module as a node, a particular connection pattern in the middle stage of the Clos network can be represented by a regular bipartite multigraph with node degree  $m$  as illustrated in Figure 12.20, where each central module corresponds to a group of  $n$  edges, each connecting one distinct pair of input–output nodes (modules).

Suppose the routing algorithm of the Clos network is based on dynamic cell switching, and the amount of traffic from input module  $I_i$  to output module  $O_j$  is  $\lambda_{ij}$  cells per time slot.



**Figure 12.20** Correspondence between the middle-stage route scheduling in a three-stage Clos network (a) and the edge-coloring of the equivalent regular bipartite multigraph (b). (©1997 IEEE.)

The connection pattern will change in every time slot according to arrival packets, and the routing will be calculated on a slot-by-slot basis. Let  $e_{ij}(t)$  be the number of edges from  $I_i$  to  $O_j$  of the corresponding bipartite multigraph in time slot  $t$ . Then the capacity  $C_{ij}$  of the virtual path between  $I_i$  and  $O_j$  must satisfy

$$C_{ij} = \lim_{T \rightarrow \infty} \frac{\sum_{t=1}^T e_{ij}(t)}{T} > \lambda_{ij}. \tag{12.5}$$

On the other hand, the routing of a circuit switched Clos network is fixed, and the connection pattern will be the same in every time slot. The capacity satisfies

$$C_{ij} = e_{ij}(t) = e_{ij} > \lambda_{ij}. \tag{12.6}$$

which implies that the peak bandwidth  $C_{ij}$  is provided for each virtual circuit at call set-up time, and it does not take the statistical multiplexing into consideration at all. We conceived the idea of quasi-static routing, called path switching, using a finite number of different connection patterns in the middle stage repeatedly, as a compromise of the above two extreme schemes. For any given  $\lambda_{ij}$ , if  $\sum_i \lambda_{ij} < n \leq m$ , and  $\sum_j \lambda_{ij} < n \leq m$ , we can always find a finite number,  $f$ , of regular bipartite multigraphs such that

$$\frac{\sum_{t=1}^f e_{ij}(t)}{f} > \lambda_{ij}, \tag{12.7}$$

where  $e_{ij}(t)$  is the number of edges from node  $i$  to node  $j$  in the  $t$ th bipartite multigraph. The capacity requirement (12.5) can be satisfied if the system provides connections repeatedly according to the coloring of these  $f$  bipartite multigraphs, and these finite amounts of routing information can be stored in the local memory of each input module to avoid the slot-by-slot



computation of route assignments. The path switching becomes circuit switching if  $f = 1$ , and it is equivalent to cell switching if  $f \rightarrow \infty$ .

The scheduling of path switching consists of two steps, the capacity assignment and the route assignment. The capacity assignment is to find the capacity  $C_{ij} > \lambda_{ij}$  for each virtual path between input module  $I_i$  and output module  $O_j$ ; it can be carried out by optimizing some objective function subject to  $\sum_i C_{ij} = \sum_j C_{ij} = m$ . The choice of the objective function depends on the stochastic characteristic of the traffic on virtual paths and the quality of service requirements of connections.

The next step is to convert the capacity matrix,  $[C_{ij}]$ , into edge-coloring of a finite number,  $f$ , of regular bipartite multigraphs, each of them representing a particular connection pattern of central modules in the Clos network. An edge-coloring of a bipartite multigraph is to assign  $m$  distinct colors to  $m$  edges of each node such that no two adjacent edges have the same color. It is well-known that a regular bipartite multigraph with degree  $m$  is  $m$ -colorable [10, 11]. Each color corresponds to a central module, and the color assigned to an edge from input module  $i$  to output module  $j$  represents a connection between them through the corresponding central module.

Suppose that we choose a sufficiently large integer  $f$  such that  $fC_{ij}$  are integers for all  $i, j$ , and form a regular bipartite multigraph, called capacity graph, in which the number of edges between node  $i$  and node  $j$  is  $fC_{ij}$ . Since the capacity graph is regular with degree  $fm$ , it can be edge-colored by  $fm$  different colors [11]. Furthermore, it is easy to show that any edge-coloring of the the capacity graph with degree  $fm$  is the superposition of the edge-coloring of  $f$  regular bipartite multigraphs of degree  $m$ . Consider a particular color assignment  $a \in \{0, 1, \dots, fm - 1\}$  of an edge between input node  $I_i$  and output node  $O_j$  of the capacity graph. Let

$$a = r \cdot f + t, \quad (12.8)$$

where  $r \in \{0, 1, \dots, m - 1\}$  and  $t \in \{0, 1, \dots, f - 1\}$  are the quotient and the remainder of dividing  $a$  by  $f$ , respectively. The mapping  $g(a) = (t, r)$  from the set  $\{0, 1, \dots, fm - 1\} \rightarrow \{0, 1, \dots, f - 1\} \times \{0, 1, \dots, m - 1\}$  is one-to-one and onto, that is

$$a = a' \iff t = t' \quad \text{and} \quad r = r'.$$

That is, the color assignment  $a$ , or equivalently the assignment pair  $(t, r)$ , of the edge between  $I_i$  and  $O_j$  indicates that the central module  $r$  has been assigned to a route from  $I_i$  to  $O_j$  in the  $t$ th time-slot of every cycle. Adopting the convention in the TDMA system, each cycle will be called a frame and the period  $f$  frame size. As illustrated by the example shown in Fig. 12.21, where  $m = 3$  and frame size  $f = 2$ , the decomposition of the edge-coloring into assignment pairs guarantees that route assignments are either space interleaved or time interleaved. Thus, the relation (12.8) will be called the time-space interleaving principle.

### 12.11.1 Homogeneous Capacity and Route Assignment

For uniform traffic, where the distribution of traffic loading between input modules and output modules is homogeneous, the  $fm$  edges of each node can be evenly divided into  $k$  groups, where  $k$  is the total number of input (output) modules. Each group contains  $g = fm/k$  edges between any I/O pair, where the frame size  $f$  should be chosen properly

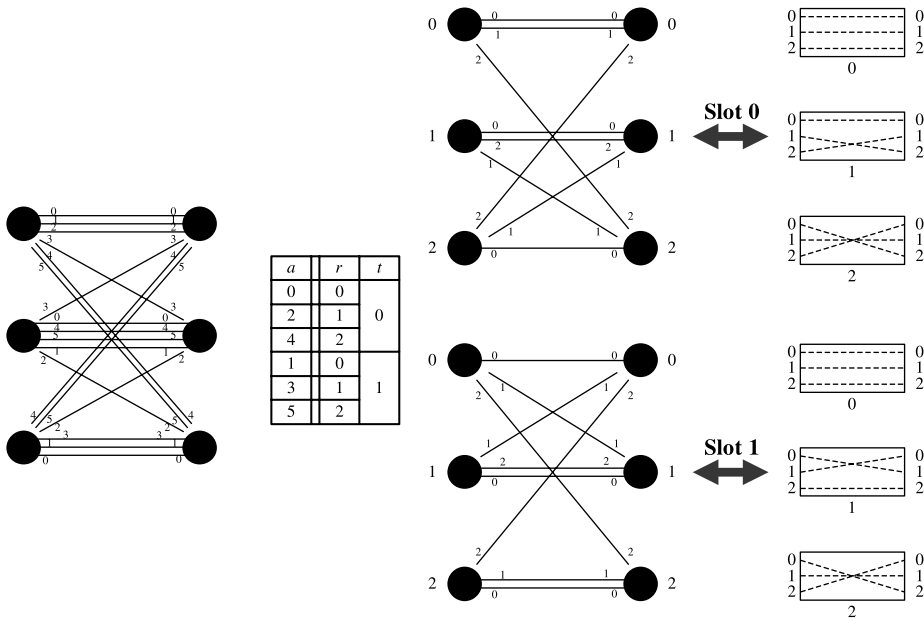


Figure 12.21 Illustration of time-space interleaving principle. (©1997 IEEE.)

to make the group size  $g$  an integer. The edges of this capacity graph can be easily colored by the Latin Square given in Table 12.1, where each  $A_i$ ,  $0 \leq i \leq k - 1$ , represents a set of distinct colors, for example,

$$A_0 = \{0, 1, \dots, g - 1\}; \quad A_1 = \{g, g + 1, \dots, 2g - 1\}; \quad \dots$$

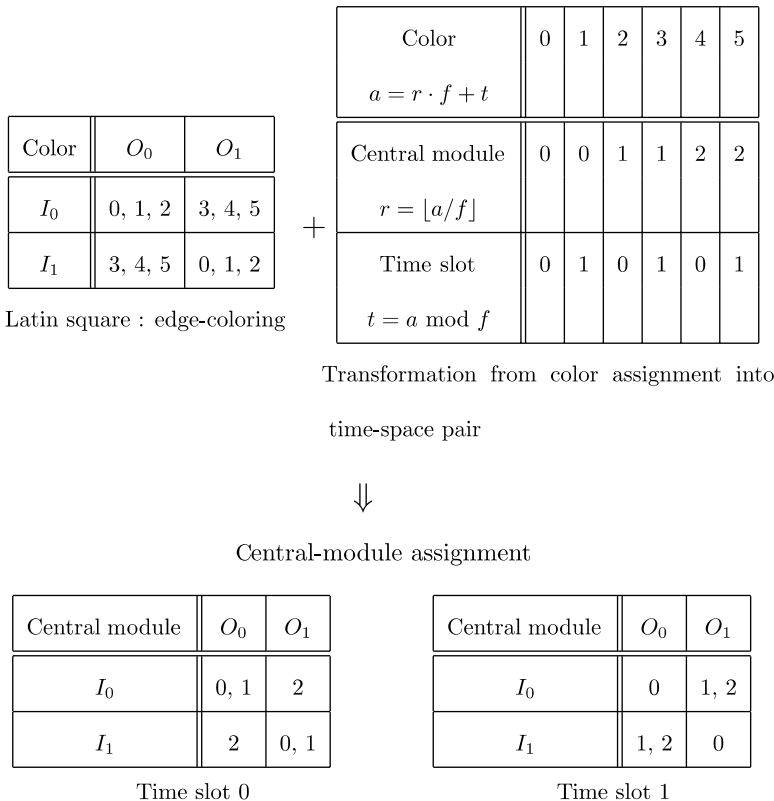
$$A_{k-1} = \{(k - 1)g, (k - 1)g + 1, \dots, kg - 1\}.$$

Since each number in the set  $\{0, 1, \dots, fm - 1\}$  appears only once in any row or column in the table, it is a legitimate edge-coloring of the capacity graph. The assignment  $a = (t, r)$  of an edge between the  $I_i/O_j$  pair indicates that the central module  $r$  will connect the input module  $i$  to output module  $j$  in the  $t$ th slot of every frame. As an example, for  $m = 3$  and  $k = 2$ , we can choose  $f = 2$  and thus  $g = 3$ .

Then, the groups of colors are  $A_0 = \{0, 1, 2\}$  and  $A_1 = \{3, 4, 5\}$ , respectively. The procedure described above is illustrated in Figure 12.22, and the correspondence between the route assignments and the connection patterns in the middle stage is shown in Figure 12.23.

TABLE 12.1 Latin Square Assignment

	$O_0$	$O_1$	$O_2$	$\dots$	$O_{k-1}$
$I_0$	$A_0$	$A_1$	$A_2$	$\dots$	$A_{k-1}$
$I_1$	$A_{k-1}$	$A_0$	$A_1$	$\dots$	$A_{k-2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$I_{k-1}$	$A_1$	$A_2$	$A_3$	$\dots$	$A_0$



**Figure 12.22** Route assignment by Latin Square for uniform traffic.

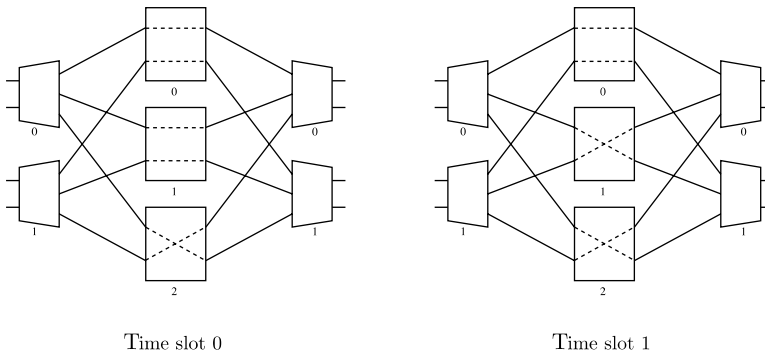
In the above example, since the number of central modules  $m$  is greater than the number of input modules  $k$ , it is possible that more than one central module is assigned to some I/O pairs in one time slot. In the case that  $m < k$ , there are not enough central modules for all I/O pairs in one time slot assignment. Nevertheless, the total number of central modules assigned to every I/O pair within a frame should be the same, for uniform input traffic to fulfill the capacity requirement, and it is equal to  $g = fm/k$ . This point is illustrated in the following example. For  $m = 4$  and  $k = 6$ , we choose  $f = 3$  and  $g = 2$ . The same method will result in the connection patterns shown in Figure 12.24. It is easy to verify that the number of central modules (paths, edges) assigned for each I/O pair is equal to  $g = 2$  per  $f = 3$  slots.

**12.11.2 Heterogeneous Capacity Assignment**

The capacity assignment in a cross-path switch is virtual-path based. It depends on the traffic load on each virtual path to allocate the capacity and determine the route assignment. The Latin Square offers a legitimate capacity assignment with homogeneous traffic, but it may not be effective anymore with heterogeneous traffic with non-uniformly distributed traffic load over the virtual paths. A more general assignment method is therefore introduced and the procedure is illustrated in Figure 12.25. The assignment procedure has four steps, each of which will be explained along with an example in the following subsections.

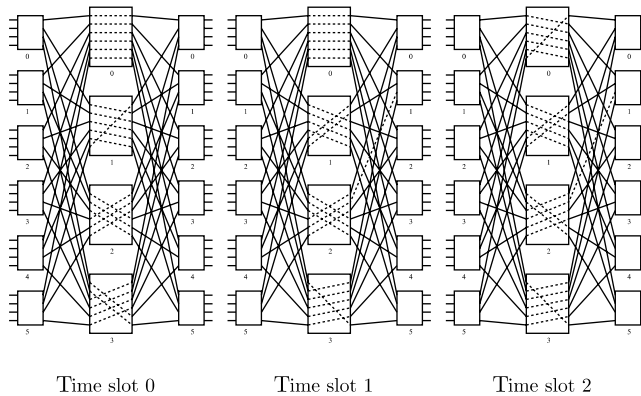
Central module	Connected $I/O$ pairs at time slot	
	0	1
0	$I_0/O_0, I_1/O_1$ (BAR)	$I_0/O_0, I_1/O_1$ (BAR)
1	$I_0/O_0, I_1/O_1$ (BAR)	$I_0/O_1, I_1/O_0$ (CROSS)
2	$I_0/O_1, I_1/O_0$ (CROSS)	$I_0/O_1, I_1/O_0$ (CROSS)

Connection pairs in central modules



**Figure 12.23** Route scheduling in the middle-stage for uniform traffic. (©1997 IEEE.)

**Virtual Path Capacity Allocation (VPCA).** This step is to allocate capacity to each virtual path based on the traffic loads. It can be formulated as an optimization problem with some traffic modeling.



**Figure 12.24** Route scheduling in central modules for the second example of uniform traffic. (©1997 IEEE.)

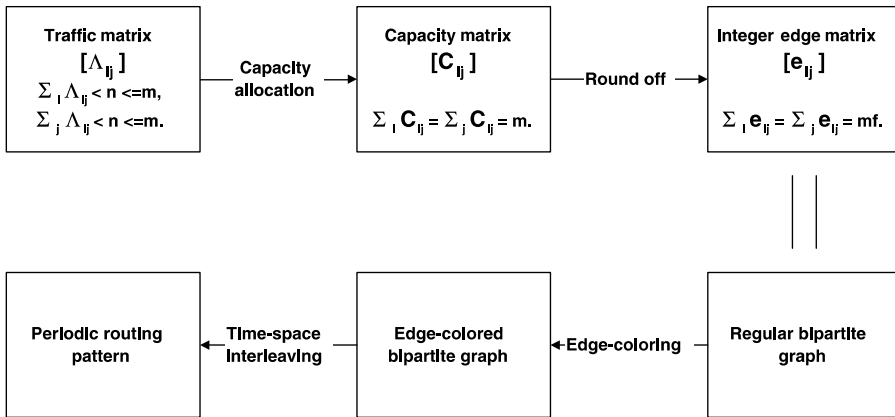


Figure 12.25 Procedure of capacity and route assignment.

Consider the cross-path switch with parameters  $n = 3$ ,  $k = 3$ , and  $m = 4$ . Suppose the traffic matrix is given by

$$\mathbf{T} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \tag{12.9}$$

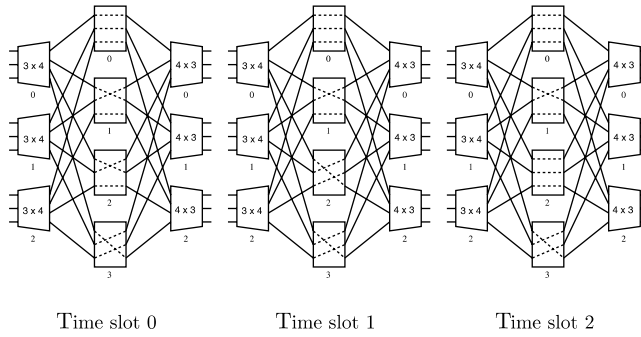
the capacity assignment matrix calculated by the minimization of input-stage delay with  $M/D/1$  model is

$$\mathbf{C} = \begin{bmatrix} 1.34 & 1.28 & 1.38 \\ 2.66 & 1.34 & 0 \\ 0 & 1.38 & 2.62 \end{bmatrix}. \tag{12.10}$$

**The Round-Off Procedure.** Some elements in the resulting capacity matrix may be non-integers. When they are rounded into integers that are required in the route assignment, round-off error arises. The concept of frame size is used to reduce the round-off error. Each element in the capacity matrix is multiplied by the frame size. Then the capacity per slot is translated into capacity per frame (see below). After that, we round the matrix into an integer matrix.

$$\mathbf{C} = \begin{bmatrix} 1.34 & 1.28 & 1.38 \\ 2.66 & 1.34 & 0 \\ 0 & 1.38 & 2.62 \end{bmatrix} \xrightarrow{\times f=3} \begin{bmatrix} 4.02 & 3.84 & 4.14 \\ 7.98 & 3.82 & 0 \\ 0 & 4.14 & 7.86 \end{bmatrix} \xrightarrow{\text{rounding}} \begin{bmatrix} 4 & 4 & 4 \\ 8 & 4 & 0 \\ 0 & 4 & 8 \end{bmatrix} = \mathbf{E}. \tag{12.11}$$

The round-off error is inversely proportional to  $f$ . That is, the error can be arbitrary small if the frame size is sufficiently large. However, since the amount of routing information stored in the memory is linearly proportional to  $f$ , the frame size is limited by the access speed and the memory space of input modules. In practice, the choice of frame size  $f$  is a



**Figure 12.26** Route Scheduling example (heterogenous traffic). (©1997 IEEE.)

compromise between the round-off error and the memory requirement. In general,

$$\mathbf{E} = \begin{bmatrix} e_{0,0} & e_{0,1} & \cdots & e_{0,k-1} \\ e_{1,0} & e_{1,1} & \cdots & e_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ e_{k-1,0} & e_{k-1,1} & \cdots & e_{k-1,k-1} \end{bmatrix} \simeq f \cdot \mathbf{C},$$

and

$$\sum_j e_{ij} = \sum_i e_{ij} = f \cdot m. \quad (12.12)$$

In the above matrix  $\mathbf{E}$ , each element  $e_{ij}$  represents the number of the edges between the input module  $i$  and output module  $j$  in the  $k \times k$  capacity graph, in which each node has a degree of  $fm$ .

**Edge-Coloring.** The capacity graph can be colored by  $fm$  colors, and each color represents one distinct time-space slot based on the time-space interleaving principle (12.8). Coloring can be found by complete matching, which is repeated recursively to reduce the degree of every node one-by-one. One general method to search for a complete matching is the so-called Hungarian algorithm or alternating-path algorithm [10, 12]. It is a sequential algorithm with the worst time complexity  $O(k^2)$ , or totally  $O(fm \times k^2)$  because there are  $fm$  matchings. If each of  $fm$  and  $k$  is a power of two, an efficient parallel algorithm proposed in [13] for conflict-free route scheduling in a three-stage Clos network with time complexity of  $O(\log^2(fmk))$  can be used. Through the time-space interleaving, the middle-stage routing pattern is obtained in Figure 12.26.

## REFERENCES

- [1] J. Hopcroft and R. Karp, "An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231 (1973).
- [2] R. Cole and J. Hopcroft, "On edge coloring bipartite graph," *SIAM Journal on Computing*, vol. 11, no. 3, pp. 540–546 (1982).

- [3] H. J. Chao, C. Lam, and E. Oki, *Broadband Packet Switching Technologies – A Practical Guide to ATM Switches and IP Routers*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2001.
- [4] H. J. Chao and S. Y. Liew, “A new optical cell switching paradigm,” in *Proc. International Workshop on Optical Burst Switching*, Dallas, Texas, pp. 120–132 (Oct. 2003).
- [5] E. Oki, Z. Jing, R. Rojas-Cessa, and H. J. Chao, “Concurrent round-robin-based dispatching schemes for Clos-network switches,” *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 830–844 (Dec. 2002).
- [6] Y. Li, S. Panwar, and H. J. Chao, “The dual round-robin matching switch with exhaustive service,” in *Proc. High Performance Switching and Routing (HPSR) 2002*, Kobe, Japan (May 2002).
- [7] N. McKeown, “iSLIP: a scheduling algorithm for input-queued switches,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201 (Apr. 1999).
- [8] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, “Low cost scalable switching solutions for broadband networking: the ATLANTA architecture and chipset,” *IEEE Communications Magazine*, vol. 35, issue 12, pp. 44–53 (Dec. 1997).
- [9] M. Karol and C.-L. I, “Performance analysis of a growable architecture for broadband packet (ATM) switching,” in *Proc. IEEE GLOBECOM’89*, Dallas, Texas, pp. 1173–1180 (Nov. 1989).
- [10] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes*. Morgan Kaufmann, San Francisco, California, 1992.
- [11] R. J. Wilson, *Introduction to Graph Theory*. Academic Press, New York, 1972.
- [12] R. J. McEliece, R. B. Ash, and C. Ash, *Introduction to Discrete Mathematics*. McGraw-Hill, New York, 1989.
- [13] T. T. Lee and S. Y. Liew, “Parallel algorithm for benes networks,” in *Proc. IEEE INFOCOM ’96*, San Francisco, California (Mar. 1996).