Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues

Richard O. LaMaire, *Member, IEEE*, and Dimitrios N. Serpanos, *Member, IEEE*

***Abstract*— We present a new scheduler, the *two-dimensional round-robin (2DRR)* scheduler, that provides high throughput and fair access in a packet switch that uses multiple input queues. We consider an architecture in which each input port maintains a separate queue for each output. In an $N \times N$ switch, our scheduler determines which of the queues in the total of N^2 input queues are served during each time slot. We demonstrate the fairness properties of the 2DRR scheduler and compare its performance with that of the *input* and *output queueing* configurations showing that our scheme achieves the same saturation throughput as output queueing. The 2DRR scheduler can be implemented using simple logic components thereby allowing a very high-speed implementation.**

I. Introduction

THERE are many applications in the computer and communications fields that require the scheduling of a system (usually a switch) that has N input resources and N output resources. This gives rise to a scheduling problem in which the resource requests can be represented by an $N \times N$ matrix RM, where $RM[R, C] = 1$ indicates that there is at least one request for the Rth input and Cth output resource pair where R and C denote row and column indexes, respectively. During a time slot, only one request can be granted in any row or column of this request matrix since a given input or output resource can only serve one request per time slot. The scheduler determines which requests are satisfied during successive time slots. The objective of the scheduler is to provide high throughput, that is, to serve as many requests as possible, while also providing fair service to the different requests. The two-dimensional round-robin (2DRR) scheduler that is described in this paper satisfies these goals. The 2DRR algorithm can be efficiently used for any such scheduling problem with N input and N output resources. Since we developed the algorithm for use in a high-speed switch, we focus on a switch application in the paper. One of the key characteristics of the 2DRR scheduling algorithm is that it allows an efficient hardware implementation so that it can be used in high-speed switches.

A related scheduling problem has been discussed in the literature. In this related problem formulation, a *traffic matrix* represents the aggregate demand for input/output pairs over a period of time slots. In a situation where the individual input queues to a switch have more than one request in them, the traffic matrix could be viewed as a tabulation of the queue contents for each input/output pair. This type of problem formulation is common in the area of satellite-switched time-division-multiple-access (SS/TDMA) systems [6]. A class of optimal algorithms (e.g., [6], [11], and references therein) have been developed to satisfy a given traffic matrix within X time slots where X is the maximum aggregate demand, from the traffic matrix, on any input or output.

Since the optimal scheduling algorithms are computationally expensive they do not lend themselves to high-speed switch applications, particularly for large N. Since the centralized *optimal* scheduler of [6] has a computation time that varies as $O(N^2)$, it is difficult to apply this scheduler to large switch sizes. To apply optimal scheduling to situations in which N is large, Rose has proposed an approach that uses N^2 processors that run algorithms whose computation times vary as $O(N)$ [11]. Thus, in high-speed packet switches with a large number of ports, the computation of the optimal schedule can require a prohibitive amount of hardware. While our approach is rather different than [11], our implementation is similar to [11] in that we use N^2 cells (i.e., a few logic gates) each of which has a run time that varies as $O(N)$. In contrast to [11], our cells are very simple and thus allow a simple high-speed implementation that scales well to large values of N.

II. Advanced Input Queueing Architecture

The switch studied in this paper uses a type of advanced input queueing in which a separate queue is maintained at each input port for each output as is shown in Fig. 1. These multiple input queues can be implemented in a shared memory for efficiency. This architecture was chosen to prevent head-of-line blocking, which occurs when a single queue is used at each input port (see [7]). Thus, in this switch, the utilization can approach unity whereas in a simple input queueing switch (i.e., one queue per input) the utilization cannot exceed 0.586 for a uniformly distributed traffic pattern and Bernoulli arrivals of unit length [7].

Alternative approaches have been taken to avoid head-of-line blocking. These approaches include window-based scheduling algorithms [5] as well as reservation schemes [10]. An approach using the framework of the SS/TDMA systems [6], [11], [4] is also provided by Chen, Liu, and Tsay [3], where an optimal time-slot assignment is calculated in an on-line fashion. In addition, Bonuccelli, Gopal and Wong [2] have considered the incremental time-slot assignment problem in which newly arrived traffic requests are scheduled optimally in a SS/TDMA satellite system. They proved that the optimal scheduling algorithm for this problem is NP-complete and

Manuscript received November 23, 1993; revised August 10, 1994; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Turner.

The authors are with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.

IEEE Log Number 9405945.

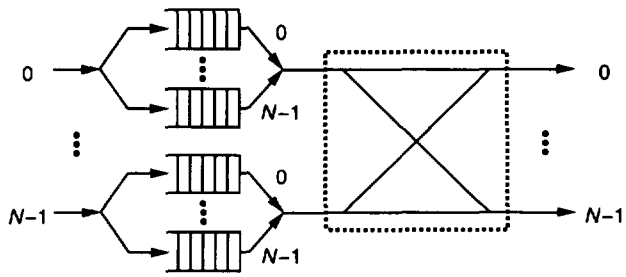


Fig. 1. Advanced input queueing architecture.

Outputs

Inputs	0	1	2	3
0	1	0	0	1
1	0	1	1	0
2	0	1	0	1
3	1	1	1	0

Fig. 2. Example request matrix.

developed several suboptimal heuristic algorithms. Karol, Eng, and Obara [8] have also developed a suboptimal algorithm for scheduling in a switch that has the advanced input queueing structure that we have described. For a uniform traffic pattern, their method of scheduling can provide an aggregate throughput of 92% without speed-up and can yield up to 95% with input grouping of size four.

The advanced input queueing switch considered in this paper operates in a slotted time-frame. That is, packets arrive at the input queues at the time slot boundaries and then, after scheduling to avoid switch contention, complete their transmissions at succeeding time slot boundaries. We assume throughout this paper that all packets require one time slot to complete their transmission.

Contention in an advanced input queueing switch is resolved through the use of a scheduling algorithm. This algorithm uses a *request matrix*, an example of which is shown in Fig. 2. In this request matrix, $RM[R, C]$, a "1" denotes a queue that has at least one waiting packet whose origin is input R and whose destination is output C, while a "0" indicates an empty queue. At each time slot, the scheduler chooses, at most, N nonconflicting requests that will actually be satisfied and produces an *allocation matrix* whose entries indicate which input/output queues can transmit in the next time slot. In this allocation matrix, the set of chosen request pairs do not use any input or output more than once.

III. The Basic 2DRR Scheduling Algorithm

We introduce a new type of scheduler that is a two-dimensional generalization of the one-dimensional round-robin scheme that is used in allocation problems with a single shared resource [9]. In this section, we describe the Basic 2DRR scheduling algorithm, while in a later section, we will describe a second version of the 2DRR scheduler that we call the Enhanced 2DRR scheduling algorithm. The Enhanced 2DRR algorithm provides improved fairness for certain traffic patterns at the cost of some additional complexity.

In an $N \times N$ switch, up to N different requests can be simultaneously served by the switch in one time slot such that no two requests are in the same row or column in the request matrix. In order to select such N elements of the request matrix, RM, we follow a method in which we examine elements of RM that belong to *generalized diagonals*.

Definition 1: A **generalized diagonal** is a set of N elements in an $N \times N$ matrix, such that no two elements are in the same row or column.

Note that there are $N!$ different generalized diagonals in an $N \times N$ matrix. In the Basic 2DRR algorithm we use only N of these diagonals by selecting one basic diagonal and then generating the remaining $N - 1$ ones by shifting the basic diagonal across the matrix (so that each matrix element is covered by one of the N diagonals). That is, by sweeping a generalized diagonal pattern of length N through the request matrix, all N^2 input/output pairs in the request matrix can be satisfied in N time slots. We use this property to guarantee a minimum amount of service to each input/output queue.

The Basic 2DRR scheduling algorithm operates in repeating cycles of N time slots in which the time slots of each cycle are indexed by the variable L, which takes on values from 0 through $N - 1$. We assume that we are given the following $N \times N$ matrices:

- **Request Matrix**: Each entry $RM[R, C]$ is binary with the semantics:

$$RM[R, C] = \begin{cases} 1, & \text{if there is at least one request for a connection from input } R \text{ to output } C; \\ 0, & \text{otherwise.} \end{cases}$$

  We use zero-based indexes for the inputs and outputs as is shown in Figs. 1 and 2.
- **Diagonal Pattern Matrix**: Each entry $DM[R, C]$ contains an integer between 0 and $N - 1$ inclusive where

$$DM[R, C] = (C - R) \bmod N. \tag{1}$$

  If $DM[R, C] = K$, then $RM[R, C]$ is covered by diagonal pattern K.
- **Pattern Sequence Matrix**: Each entry $PM[I, J]$ is an integer between 0 and $N-1$ inclusive with the semantics: $PM[I, J] = K$ implies that when the time slot index L of a cycle is equal to J, then the I-th diagonal pattern in the sequence applied by the algorithm is the one numbered K in the diagonal pattern matrix. The ordering index I varies from 0 to $N - 1$.

Using these matrices, the Basic 2DRR algorithm produces the **Allocation Matrix**, AM, with binary entries and the semantics:

$$AM[R, C] = \begin{cases} 1, & \text{if a connection is allocated from input R to output} C; \\ 0, & \text{otherwise.} \end{cases}$$

At the beginning of time slot L in a cycle, all entries of the allocation matrix are set to zero. Then a sequence of N diagonal patterns is applied to the request matrix in the order specified by the pattern sequence matrix PM. That is, the diagonal pattern with index $PM[0, L]$ is applied first followed by diagonal pattern $PM[1, L] \ldots PM[N-1, L]$. As these diagonal patterns are overlaid on the request matrix, the entry $AM[R, C]$ is set to 1 at the I-th point ($0 \leq I \leq N-1$) in the sequence if the following conditions are true.

1) $RM[R, C] = 1$.
2) Input R and output C are still available for allocation (i.e., they have not been allocated to a different connection by a previously applied diagonal in the current time slot).
3) $DM[R, C] = K$, where $PM[I, L] = K$.

The above scheduling procedure is repeated for each cycle of N successive time slots. That is, after a cycle has been completed with the use of column $N-1$ of the pattern sequence matrix, we begin the scheduling procedure over with column 0 of the pattern sequence matrix.

As was discussed above, the Basic 2DRR algorithm provides a fairness guarantee that each of the N^2 input/output queues will receive at least one opportunity for service during every cycle of N time slots. After providing this basic fairness guarantee, there is a second problem of how to *fairly* serve those requests that are not in the current diagonal, but for which resources (i.e., input or output ports) are still available after the basic fairness guarantee has been provided. In the Basic 2DRR algorithm, if the diagonal patterns of Fig. 3 were used in numerically increasing order in the scheduling sequence at every time slot, then the resulting algorithm would yield significant unfairness by always favoring the elements of a traffic pattern that were encountered first by the sweeping diagonal. In our studies of this problem, we have found this situation to be particularly apparent when the traffic pattern in the request matrix is line-shaped. To solve this problem, we seek a pattern sequence in which no pattern index is consistently favored over the other indexes. With this goal in mind, we consider the pattern sequence matrix that is shown at the top of Fig. 3. As is shown in the figure, a different ordering of all diagonals is applied at different time slots. Note that a different one of the N patterns is applied *first* during each time slot to provide the aforementioned basic fairness guarantee. Further, note that if the diagonal patterns are applied according to the pattern sequence matrix of Fig. 3, then over a period of $N = 4$ time slots, no pattern receives sequencing preference.

To illustrate the operation of the Basic 2DRR algorithm, we show in the lower part of Fig. 3 an example for the request matrix of Fig. 2. In time slot 0, pattern 0 of the diagonal pattern matrix is applied resulting in the granting of request pairs (0,0) and (1,1) and hence the allocation of inputs 0 and 1 and outputs 0 and 1. Since some inputs and outputs are still unallocated, we apply the next diagonal, pattern 1, as indicated by column 0 of the pattern sequence matrix 0. In this case, the request pair (2,3) is granted. Note that request pairs (1,2) and (3,0) could not be granted at this step because input 1 and output 0, respectively had already been allocated. After patterns 2 and 3 have been applied in time slot 0, the resulting allocation of four request pairs for this time slot is shown. A similar procedure is used for time slot 1, however, in this case the pattern sequence of column 1 is now used.

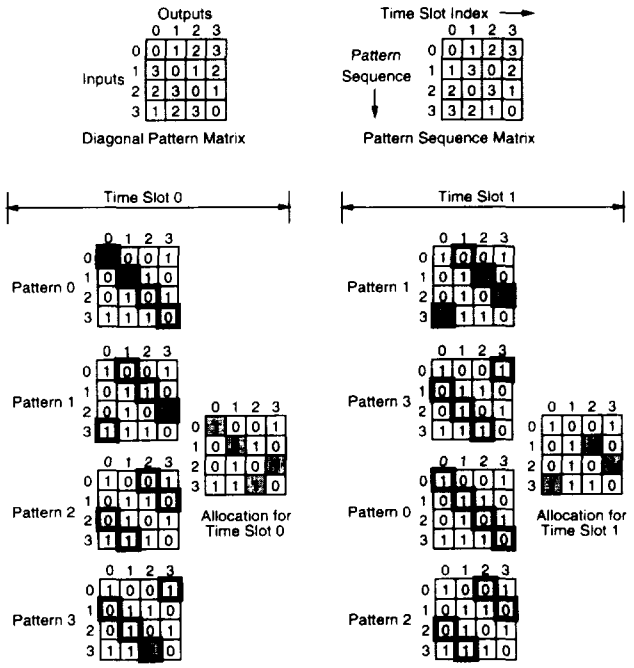


Fig. 3. Operation of the Basic 2DRR scheduling algorithm over two time slots.

The algorithm for generating the pattern sequence matrix of the Basic 2DRR algorithm is shown in Algorithm 1 at the top of the next page. In this algorithm, the *modulus* operator can be implemented in the following simple form since we know that $V + OFFSET < 2M - 1$ or $Q < 2S - 1$ in the following:

$$modulus(Q, S) = \begin{cases} Q, & \text{if } Q < S \\ Q - S, & \text{otherwise.} \end{cases}$$

We note that the maximum number of times that the body of the innermost do-loop executes is given by the integer part of $\frac{M-N}{2}$ plus 1 (i.e., $\lfloor \frac{M-N}{2} \rfloor + 1$). Thus, when $N+1$ is prime, the do-loop body is only executed once and can be replaced by a single statement in these cases.

The pattern sequence matrix PM that results from Algorithm 1 can be expressed in the following iterative form for $1 \leq I \leq N-1$:

$$PM[I, J] = \begin{cases} (PM[I-1, J] + J + 1) \bmod M, \\ \quad \text{if } PM[I-1, J] < N - (J+1), \\ \left(PM[I-1, J] + \left\lceil \frac{M - PM[I-1, J]}{J+1} \right\rceil (J+1)\right) \bmod M, \\ \quad \text{if } N - (J+1) \leq PM[I-1, J]. \end{cases} \quad (2)$$

where $PM[0, J] = J$ for $0 \leq J \leq N-1$. Further, if $N+1$ is prime, in which case $M = N+1$, it can be shown that

$$PM[I, J] = (I + IJ + J) \bmod (N+1). \quad (3)$$

```
do from J = 0 until J = N − 1 {          /* sequence through N columns */
    OFFSET = J + 1
    V = −1                               /* setup to make J first pattern in sequence */
    do from I = 0 until I = N − 1 {      /* sequence through N rows */
        do {                             /* compute next pattern number in sequence */
            V = modulus(V + OFFSET, M)
        } until (V < N)                  /* ignore the values N, ..., M − 2 */
        PM[I, J] = V                     /* store the pattern sequence */
    } end_do
} end_do
```

Algorithm 1. For computing the pattern sequence matrix, $PM[0...N-1, 0...N-1]$, where the modulus M is the smallest prime number that is greater than or equal to $N+1$.

IV. Analysis Of The Basic 2DRR Algorithm

In this section, we consider the fairness, throughput and delay properties of the Basic 2DRR algorithm. The fairness properties of the Basic 2DRR algorithm are determined by the properties of the diagonal pattern matrix and the pattern sequence matrix. Our basic fairness guarantee is given by the following.

Theorem 1: For the Basic 2DRR scheduling algorithm, over every period of N time slots, each input/output queue (i.e., each element of the request matrix, $RM[R, C]$) receives at least one time slot of service.

Proof: The N diagonal patterns, $0, 1, \ldots, (N-1)$, in the matrix of (1) cover all N^2 elements of an $N \times N$ request matrix. Further, since the 0-th row of the pattern sequence matrix contains all N diagonal pattern indexes, 0 through $N-1$, we conclude that over every N time slot period, each request element (i.e., each input/output queue) will have an opportunity to transmit at least one of any waiting packets. Q.E.D. □

The pattern sequence matrix of Fig. 3 has some important properties that help provide fair scheduling in the Basic 2DRR algorithm. In preparation, we define the following properties:

1) **Maximum Direct Ordering:** The maximum, for all index pairs, of the number of times that the two diagonal pattern indexes follow one another in the pattern sequence matrix. If this property is greater than one, then some indexes receive preference in sequencing over others.
2) **Maximum Row(Column) Frequency:** The maximum, for all indexes, of the maximum number of times that the diagonal pattern index appears in any one row (column) of the pattern sequence matrix. If this property is greater than one, then (on average) some indexes are tried in sequence earlier than others across the N time-slot cycle of the Basic 2DRR algorithm.

As an illustration of these properties we see that, in Fig. 3, no pattern index follows any other index more than once. For example, the column sequence 0-1 only appears once in the matrix. Thus, the maximum direct ordering is one for this pattern sequence matrix. Further, in any row of the pattern sequence matrix of Fig. 3, each index appears only once, so the maximum row frequency is also one. Using the above definitions, we can prove the following theorem.

Theorem 2: If $N+1$ is prime, then the maximum direct ordering is one and the maximum row frequency is one for the pattern sequence matrix that is defined by Algorithm 1.

Proof: If $N+1$ is prime, then M is equal to $N+1$ by the definition of M in Algorithm 1. Further, if we assume that $V < N$ in each step of Algorithm 1, then by expanding the iteration with the initial condition of $PM[0, J] = J$ we find that

$$PM[I, J] = (I + IJ + J) \bmod (N+1), \quad 0 \le I \le N-1,\ 0 \le J \le N-1. \tag{4}$$

We verify that $V < N$ in each step when $N+1$ is prime by noting that if V was equal to N in a step then

$$I + IJ + J = N \tag{5}$$

and

$$(I+1)(J+1) = N+1. \tag{6}$$

Since $N+1$ is prime and $I < N$ and $J < N$, (6) is a contradiction and therefore $V < N$ in each step of Algorithm 1. Having verified (4) we use it to show that both the maximum direct ordering and the maximum row frequency are one when $N+1$ is prime.

To show that the maximum row frequency is one, we must show that

$$PM[I, J] \neq PM[I, K], \text{ for } 0 \le K \le N-1 \text{ where } K \neq J. \tag{7}$$

Using (4), (7) is equivalent to

$$(I + IJ + J) - (I + IK + K) \neq Q(N+1), \tag{8}$$

where Q is an integer. This in turn is equivalent to

$$(I+1)(J-K) \neq Q(N+1). \tag{9}$$

Since $1 \le I+1 \le N$, $-(N-1) \le J-K \le N-1$, $K \neq J$ and $N+1$ is prime, we conclude that there is no integer Q that satisfies (9). Thus, we have proven that if $N+1$ is prime, then the maximum row frequency is one.

Next, we prove that the primeness of $N+1$ also implies that the maximum direct ordering is one. We assume that two different elements, $PM[I_1, J_1]$ and $PM[I_2, J_2]$, both take on the same value, S. From Algorithm 1 and the earlier part of this proof, in which we proved that $V < N$, we conclude that

$$PM[I, J] = (PM[I-1, J] + J + 1) \bmod (N+1), \quad 1 \le I \le N-1,\ 0 \le J \le N-1. \tag{10}$$

Thus, the elements that follow $PM[I_1, J_1]$ and $PM[I_2, J_2]$ are given by

$$(S + J_1 + 1) \bmod (N+1),\ 0 \le S \le N-1,\ 0 \le J_1 \le N-2, \tag{11}$$

and

$$(S + J_2 + 1) \bmod (N+1),\ 0 \le S \le N-1,\ 0 \le J_2 \le N-2, \tag{12}$$

respectively. For these two elements to be equal, it must be true that

$$J_1 - J_2 = Q(N+1), \text{ where } Q \text{ is an integer.} \tag{13}$$

Since $-(N-2) \le (J_1 - J_2) \le N-2$ and $J_1 \ne J_2$ (see Corollary 1), we conclude that (13) is not true. Thus, we have proven that if $N+1$ is prime, then the elements that follow a value S have distinct values in each of the $N-1$ columns in which S is not the last column element. That is, if $N+1$ is prime, then the maximum direct ordering is one. Q.E.D. □

Remark 1: From (4), it is clear that if $N+1$ is prime, then the pattern sequence matrix yielded by Algorithm 1 is symmetric. In this case, only $\frac{N(N+1)}{2}$ elements of the $N \times N$ pattern sequence matrix need to be computed.

Corollary 1: The maximum column frequency is one for the pattern sequence matrix that is defined by Algorithm 1. That is, each of the indexes $0 \ldots N-1$ appears once and only once in each column.

Proof: When $N+1$ is prime, Theorem 2 tells us that the maximum row frequency is one which implies that the N entries in each row of the pattern sequence matrix are distinct. Since the pattern sequence matrix is symmetric when $N+1$ is prime, we conclude that the N entries in each column of the pattern sequence matrix are distinct so that the maximum column frequency is one.

Now, we consider the case when $N+1$ is not prime. In this case, we can show that the pattern sequence matrix produced by Algorithm 1 corresponds to taking a larger pattern sequence matrix for the case of $\hat{N} = M-1$ and omitting: 1) columns with indexes N through $\hat{N}-1$ inclusive, and 2) the entries in the remaining columns that have values of N through $\hat{N}-1$ inclusive. The column entries are compacted upward to remove any gaps that are created by omission 2. Thus, the fact that the $\hat{N}$ entries in each column of the pattern sequence matrix for the $\hat{N}+1$ case are distinct implies that the N entries in each column of the pattern sequence matrix for the N case are also distinct so that the maximum column frequency is one. Q.E.D. □

While Theorem 2 shows us that no diagonal pattern index receives preferred treatment when $N+1$ is prime, we cannot make this same claim when $N+1$ is not prime. However, when $N+1$ is not prime, we attempt to come close to the desired properties of Theorem 2 by using a value of M that is the smallest prime that is greater than $N+1$. In Table I, we investigate the properties of the pattern sequence matrix for $N \le 50$. Note, in Table I, that when $N+1$ is prime, the maximum direct ordering and maximum row frequency numbers are both one, as expected. The extent to which our pattern sequence matrix comes close to achieving the ideal properties specified in Theorem 2 depends on how close $N+1$ is to M, the smallest prime number that is greater than or equal to $N+1$. That is, in Table I, the value of both the maximum direct ordering and the maximum row frequency are monotonically decreasing as the difference between $N+1$ and M decreases.

TABLE I
PROPERTIES OF THE PATTERN SEQUENCE MATRIX FOR DIFFERENT SWITCH SIZES

N	*Max. Direct Ordering*	*Max. Row Frequency*	N	*Max. Direct Ordering*	*Max. Row Frequency*
1	-	-	26	2	3
2	1	1	27	2	2
3	2	2	28	1	1
4	1	1	29	2	2
5	2	2	30	1	1
6	1	1	31	3	5
7	3	4	32	3	4
8	2	2	33	3	4
9	2	2	34	2	3
10	1	1	35	2	2
11	2	2	36	1	1
12	1	1	37	3	4
13	3	4	38	2	3
14	2	3	39	2	2
15	2	2	40	1	1
16	1	1	41	2	2
17	2	2	42	1	1
18	1	1	43	3	4
19	3	4	44	2	3
20	2	3	45	2	2
21	2	2	46	1	1
22	1	1	47	3	4
23	3	4	48	3	4
24	3	4	49	3	4
25	3	4	50	2	3

Returning to our earlier example, in Fig. 3, we see that $N+1 = 5$ is prime, so for this case the maximum direct ordering and the maximum row frequency are both one. However, when $N+1$ is not prime as is the case in Fig. 4 where $N+1 = 4$, both the maximum direct ordering and the maximum row frequency are equal to two. This results in a slight unfairness, in that certain diagonal pattern indexes receive sequencing preference relative to other indexes. For example, in Fig. 4, the index 0 receives some preference at the expense of index 2 because diagonal pattern 0 is applied, on average, earlier in sequence than diagonal pattern 2.

We now consider the saturation throughput of the Basic 2DRR scheduling algorithm. Consider a situation in which all elements of the $N \times N$ request matrix continually have a backlog of requests in them. In this case, the Basic 2DRR algorithm will serve all N elements in each of the N diagonal patterns over the repeating N time slot cycle of the Basic 2DRR algorithm. Thus, the advanced input queueing structure and the Basic 2DRR scheduling algorithm avoid head-of-

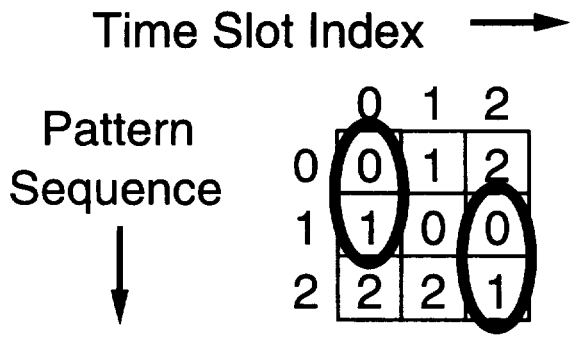


Fig. 4. An example pattern sequence matrix when $N + 1$ is not prime.

line blocking and achieve a saturation throughput of one for all input and output links in this uniform traffic pattern case.

To examine the delay characteristics of the Basic 2DRR algorithm, we use the well-known case of a uniform traffic pattern with Bernoulli arrivals. Specifically, we assume that unit-length packets arrive on the N input links according to independent and identically distributed Bernoulli processes. That is, the probability that a packet will arrive on a given input link is p. A packet has an equal probability of being destined for each of the N outputs. For comparison, we consider the cases of pure input queueing and output queueing (see [7]). For the input queueing case, one can use random scheduling (see [7]) when a conflict arises between multiple input queues that have a head-of-line packet with the same output destination. Alternatively, a one-dimensional round-robin (1DRR) scheduling algorithm can be used. In our implementation of this well-known scheduling policy, we used an N-element storage vector to keep track of the input queue that was last served by each of the N output links. At a time slot, for a given output, we scan the input queues (modulo N) looking for a head-of-line packet that seeks the given output. We begin this scanning with the input queue that follows the one served during the last time slot in which a service occurred for this output. In this way, no input queue is preferred over others in the resolution of output conflicts.

In Fig. 5, we show the mean waiting time (not including the packet transmission time of one slot) for an 8×8 switch in which the following queue structures and scheduling types were used: 1) input queueing with random and 1DRR scheduling, 2) advanced input queueing with Basic 2DRR scheduling, and 3) output queueing. These simulation results were produced by using very long runs (many batches) that yielded very small confidence intervals. As can be seen from Fig. 5, the random and 1DRR scheduling yield nearly identical mean waiting times for the input queueing case. For the input queueing case of $N = 8$ with random scheduling, a saturation throughput of 0.618 has been observed [7]. (Note that the saturation throughput as $N \rightarrow \infty$ is 0.586.) For both the advanced input queueing case with Basic 2DRR scheduling and the output queueing case, the saturation throughput is one. However, the output queueing case has a lower mean waiting time, but this approach requires the internal switching fabric to have a speed-up of N as compared with that of the input and advanced input queueing approaches.

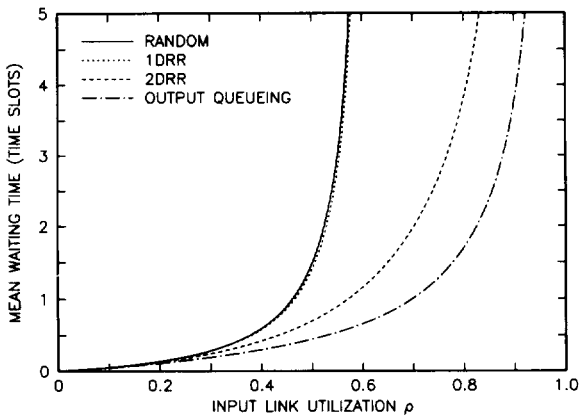


Fig. 5. Mean waiting time for a uniform traffic case and an 8×8 switch.

Outputs

Inputs	0	1	2	3	4	5	6	7
0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0
2	0	0	0	0	1	0	0	0
3	1	1	1	1	1	1	1	1
4	0	0	0	0	1	0	0	0
5	0	0	0	0	1	0	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	1	0	0	0

Fig. 6. Request matrix for cross-shaped traffic case.

It is also interesting to consider the case of having traffic hot-spots. Consider the cross-shaped traffic pattern of Fig. 6. The arrivals on the input links are independent and identically distributed Bernoulli processes. Packets arrive, with equal probability, on input links 0 through 7, but all packets arriving on input links 0 through 2 and 4 through 7 are destined for output link 4, whereas each packet arriving on input link 3 is destined with equal probability for one of the 8 outputs. We have found this pattern to be particularly stressing of the fairness properties of schedulers. In a single switch network, this pattern is not likely to arise, since the center input/output pair of the cross would correspond to a connected entity sending packets to itself. However, for multiple switch networks, input and output ports would not necessarily correspond to the same entity and this traffic pattern could arise. This is particularly true of client/server applications, where multiple requests are targeted to the server, which in turn transmits to many clients.

We compare the mean waiting time results for input queueing with 1DRR scheduling and advanced input queueing with Basic 2DRR scheduling. For the (0,4) input/output request pair, the 1DRR and 2DRR scheduling algorithms yield similar mean waiting times. The 2DRR result for the (0,4) pair is slightly larger than the 1DRR result for this pair since in the 2DRR case, the central (3,4) element receives fairer treatment (i.e., a greater portion of output link 4's utilization). The most dramatic effect is the increased throughput and greatly

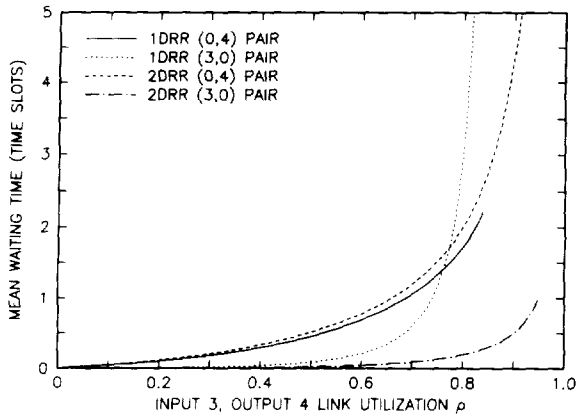


Fig. 7. Mean waiting time for cross-shaped traffic case.

decreased delay that is provided to the (3,0) pair with 2DRR as compared with 1DRR and input queueing. This performance improvement is the result of the advanced input queueing and Basic 2DRR scheduling overcoming the head-of-line blocking effect that occurs in the input queueing case for packets that arrive on input link 3. Note that input/output pair (3,4) obtains service for a fraction of time that is no more than $1/N$ due to contention for output 4. Thus, for the input queueing case, packets from input 3 to output 4 frequently block other packets in the input 3 queue.

V. The Enhanced 2DRR Scheduling Algorithm

The Basic 2DRR scheduling algorithm uses a total of N diagonal patterns in an attempt to provide fair scheduling. There are actually $N!$ different diagonal patterns that can be considered for a $N \times N$ request matrix. We will show that the fairness properties of the Basic 2DRR scheduling algorithm can be improved by using a larger number of diagonal patterns than N. In this section, we introduce the Enhanced 2DRR scheduling algorithm which uses a total of N^2 different diagonal patterns.

The Enhanced 2DRR scheduling algorithm uses N different diagonal pattern matrices (each of which is comprised of N diagonal patterns), but still uses the same pattern sequence matrix that is used in the Basic 2DRR scheduling algorithm. As was shown in Fig. 3, the patterns in the diagonal pattern matrix of the Basic 2DRR scheduling algorithm are generated by shifting the main diagonal. Thus, the diagonal pattern that is composed of the elements (0,0), (1,1), ... $(N-1, N-1)$, is the *generator* of the diagonal pattern matrix of the Basic 2DRR scheduling algorithm. In the Enhanced 2DRR algorithm, N different *generators* are used to generate the N different diagonal pattern matrices. These generating diagonal patterns are derived from the pattern sequence matrix by applying a new interpretation to the meaning of the matrix. We interpret the row index of the pattern sequence matrix as corresponding to the input link index as is done in the request matrix. Further, we interpret the entries of the pattern sequence matrix as column indexes of a diagonal pattern, so that each column of the pattern sequence matrix represents a different diagonal. To illustrate the operation of the Enhanced 2DRR scheduling algorithm, we show in Fig. 8, the $N = 4$ diagonal pattern

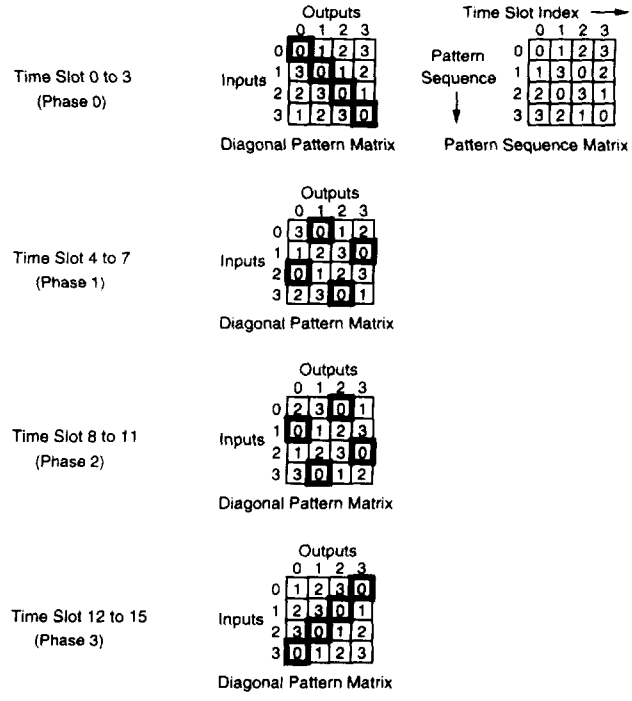


Fig. 8. Operation of the Enhanced 2DRR scheduling algorithm, where the four *generating* diagonal patterns are shown in bold boundaries for each of the different diagonal pattern matrices that are used.

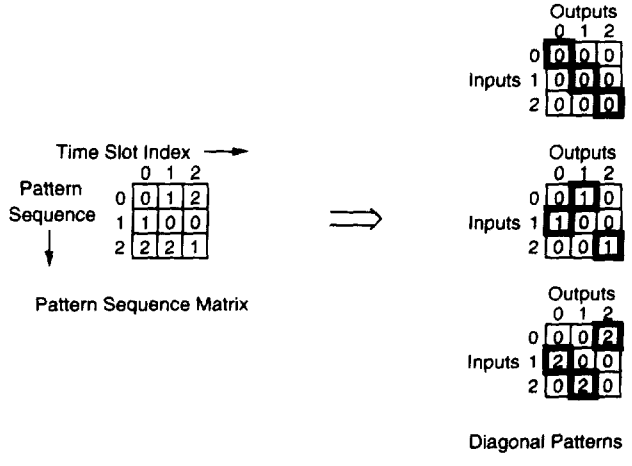


Fig. 9. Example of the *generators* that result from the pattern sequence matrix when $N + 1$ is not prime. Note that the (1,2) and (2,0) request pairs are not covered while the (1,0) and (2,2) request pairs are double covered by the three diagonal patterns.

matrices that are used during N phases of N time slots each. The generators of these diagonal pattern matrices are the elements that are marked with bold boundaries. Note how diagonal patterns 1, 2, and 3 are obtained by shifting the generating diagonal pattern to the right. The same pattern sequence matrix is used for each diagonal pattern matrix. When $N+1$ is not prime, the N generators can still be obtained from the pattern sequence matrix, but now the generators do not have the property that they provide a complete covering of the N^2 elements of the request matrix (see Fig. 9). This is not a problem for the Enhanced 2DRR algorithm since it does N shifts in each stage to cover all of the N^2 request elements.

The Enhanced 2DRR scheduling algorithm operates in repeating cycles of N^2 time slots in which the time slots of each cycle are indexed by the variable L which takes on values from 0 through N^2-1. The Enhanced 2DRR algorithm can be viewed as applying the Basic 2DRR algorithm in N different phases, where a different diagonal pattern matrix is used during each phase of N time slots. The main steps of the Enhanced 2DRR scheduling algorithm are as follows.

1) Define the N diagonal pattern matrices, labeled matrix 0 through $N-1$, and the pattern sequence matrix that will be used.
2) In the first scheduling phase, which lasts from time slot 0 to $N-1$, use diagonal pattern matrix 0 and the pattern sequence matrix as was described in the Basic 2DRR algorithm.
3) In the second phase, which lasts from time slot N to $2N-1$, use diagonal pattern matrix 1 and the same pattern sequence matrix.
4) Continue the procedure of steps 2) and 3) for the remaining phases and diagonal pattern matrices until phase $N-1$ is completed at time slot N^2-1. At this time, the entire process that begins with step 2) is repeated.

We can formalize this procedure by defining a set of N different diagonal pattern matrices, $DM_P[R,C]$, one for each phase P as follows:

$$DM_P[R,C] = (C - PM[R,P]) \bmod N,$$
$$\text{where } P = \left\lfloor \frac{L}{N} \right\rfloor, 0 \le L \le N^2 - 1. \tag{14}$$

VI. Analysis of the Enhanced 2DRR Algorithm

In this section, we consider the fairness, throughput and delay properties of the Enhanced 2DRR scheduling algorithm and compare it with the properties of the Basic 2DRR scheduling algorithm. Since different diagonal pattern matrices are used for consecutive sets of time slots, the Enhanced 2DRR algorithm has a looser fairness guarantee than the Basic 2DRR algorithm.

Theorem 3: For the Enhanced 2DRR scheduling algorithm: a) over every period of $2N-1$ time slots, each input/output queue (i.e., each element of the request matrix, $RM[R,C]$) receives at least one time slot of service; b) over every period of N^2 time slots, each input/output queue receives at least N time slots of service for an average of one time slot of service for every N time slots.

Proof: For each diagonal pattern matrix DM_P ($0 \le P \le N-1$), the N diagonal patterns cover all N^2 elements of an $N \times N$ request matrix. Since two consecutive phases of the Enhanced 2DRR algorithm use two different diagonal pattern matrices, we must consider the relationship of the these two matrices. In the worst case situation, one diagonal pattern matrix will cover a given input/output queue in diagonal pattern 0 and the diagonal pattern matrix of the next phase will cover the given input/output queue in diagonal pattern $N-1$. In this case, the worst case time between services will be $2N-1$ which implies a). Now, to prove b), we note that since the 0-th row of the pattern sequence matrix contains all N diagonal pattern indexes, 0 through $N-1$, we conclude that over every N^2 time slot period, each request element (i.e., each input/output queue) will have an opportunity to transmit at least N of any waiting packets. Q.E.D. □

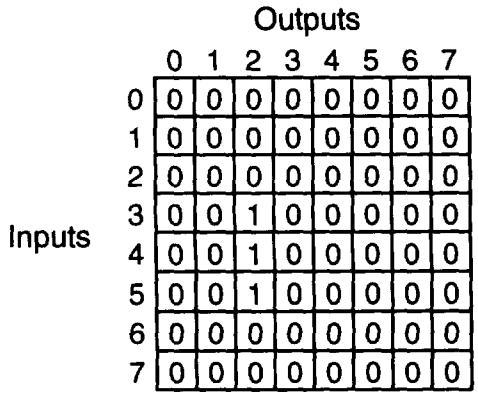


Fig. 10 Request pattern for line-shaped traffic case.

The saturation throughput of the Enhanced 2DRR scheduling algorithm for a uniform traffic pattern is one since it is really just N phases of Basic 2DRR type of scheduling. In the next example, our motivation for developing the Enhanced 2DRR scheduling algorithm becomes clear. We consider the vertical line-shaped traffic pattern of Fig. 10. To compute the saturation throughputs for this pattern, we assume that requests are always present for the three input/output pairs that are indicated with a one in Fig. 10. In this case, we obtain the throughput results of Fig. 11 for the Basic and Enhanced 2DRR algorithms. We have considered many types of traffic patterns in our investigation of the properties of these two scheduling algorithms. For the example of Fig. 10, the Basic 2DRR algorithm provides somewhat unequal throughputs to the three input/output pairs [see Fig. 11(a)]. If the line-shaped pattern included all $N = 8$ of the input/output pairs in the column for output 2 (see Fig. 10), then each one would have a saturation throughput of $\frac{1}{N} = 0.125$ since each pair is guaranteed to receive one time slot of service during every period of N time slots. In the reduced length line of Fig. 10, the (4,2) pair receives its basic throughput guarantee of 0.125, but the other two elements receive greater amounts of throughput. For the five elements of column 2 that are zero, the pattern sequence matrix does not serve a request using the diagonal pattern index that is in the 0th row of the pattern sequence matrix. In these five cases, the pattern sequence matrix is used to sequence through the diagonal patterns until a pattern hits one of the three elements of the line-shaped traffic pattern. As can be seen, the (5,2) pair is the most common hit that occurs during this procedure. In the Enhanced 2DRR algorithm, we use a larger number of diagonals, N^2 as opposed to N for the Basic 2DRR algorithm, which tend to hit the three elements of the line-shaped pattern in a more even manner over the N^2 time-slot cycle of the Enhanced 2DRR algorithm. Thus, the throughputs that are shown in Fig. 11(b) are closer to being equal than in the Basic 2DRR case.

We now consider the delay performance of the Enhanced 2DRR algorithm. For the uniform traffic pattern case that

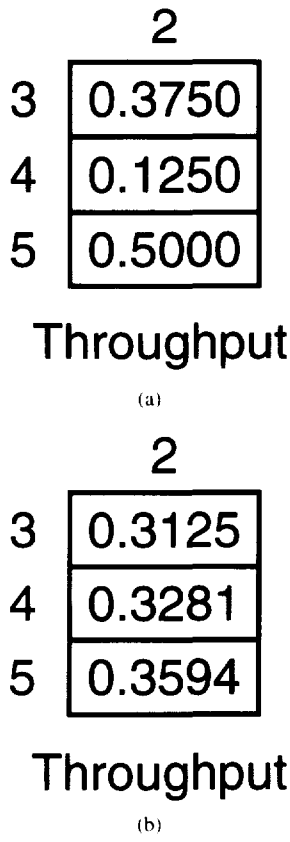


Fig. 11. Saturation throughputs for each input/output pair. (a) Basic 2DRR. (b) Enhanced 2DRR.

was discussed in Section IV, the Enhanced 2DRR algorithm yields results that are identical to those of the Basic 2DRR algorithm (see Fig. 5). Further, for cases in which the traffic patterns occupy an entire row and/or column, such as the cross-shaped traffic pattern of Fig. 6, the Basic and Enhanced 2DRR algorithms yield similar delay performance. However, in cases in which the traffic patterns do not fill entire lines in at least one dimension, the delay results of the two algorithms can differ significantly, as is suggested by the throughput results of Fig. 11.

Consider again the line-shaped traffic pattern of Fig. 10. To examine delay properties, we assume that the arrivals on the input links are independent and identically distributed Bernoulli processes. Packets arrive, with equal probability, on input links 3–5 and are all destined for output link 2. In Figs. 12 and 13, we show the mean waiting time results for the Basic and Enhanced 2DRR algorithms, respectively. As expected from the throughput results for the Basic 2DRR algorithm, we see that the mean waiting times for the three input/output pairs differ greatly from their composite mean. In contrast, in the Enhanced 2DRR case, all three input/output pairs have similar mean waiting times. That is, the Enhanced 2DRR algorithm is fairer for these types of traffic patterns. We note that the composite mean waiting time is the same for both algorithms.

VII. Implementation

One of the key characteristics of the Basic and Enhanced 2DRR algorithms is that they can be easily implemented

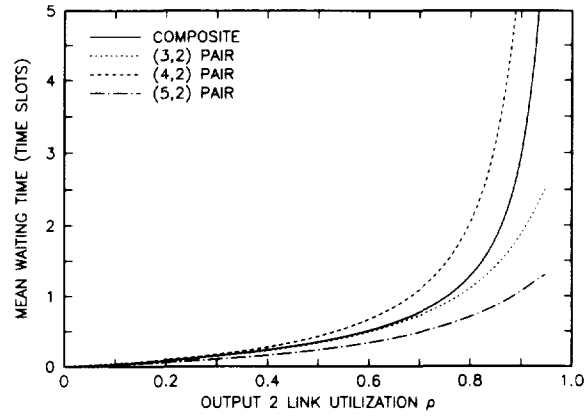


Fig. 12. Mean waiting time for Basic 2DRR algorithm and line-shaped traffic case.

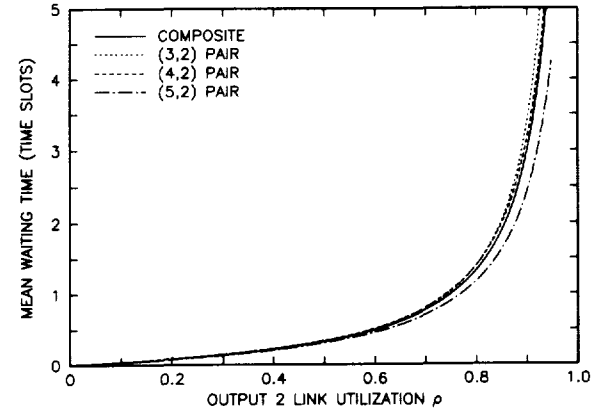


Fig. 13. Mean waiting time for Enhanced 2DRR algorithm and line-shaped traffic case.

in hardware and thus incorporated in high-speed switches. In the following section we describe the architecture and design of a high-speed scheduler that implements the Basic 2DRR algorithm as well as the architecture of a scheduler for the Enhanced 2DRR algorithm. The characteristics of a prototype implementation of the Basic 2DRR scheduler are also discussed.

A. Basic 2DRR Scheduler

Fig. 14 shows the architecture of the Basic 2DRR design for an $N \times N$ switch as well as the clocks required for the system's operation in the case of a 4×4 switch. The *time-slot clock* (TSC) is the switch clock, i.e. a packet is transferred from an input to an output during one period of the TSC clock. The *interval clock* (IC) has a frequency that is N (in this case 4) times that of the TSC clock, and regulates the application of the diagonals within a time slot. The use of the clocks in the calculation of the granted transfers can be simply described as follows: during each time slot (one TSC period), a column of the pattern sequence matrix is used; within a TSC cycle, the N IC cycles are used to apply the N diagonals, one during each IC cycle.

The scheduler uses as input the transmission requests, i.e., the data of the *request matrix* (RM) as Fig. 14 shows.

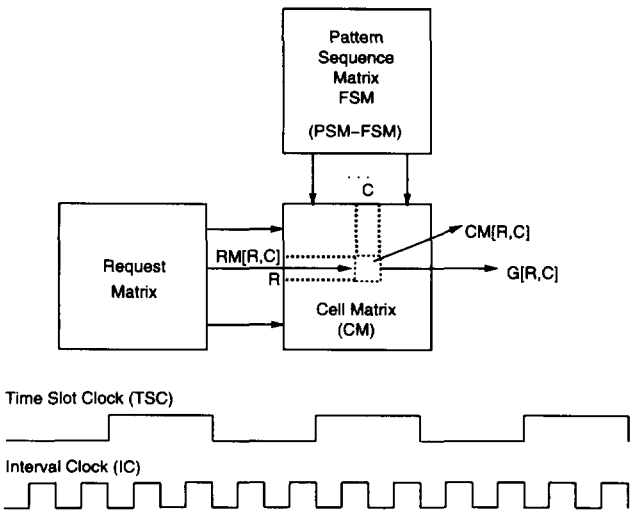


Fig. 14 2DRR design.

$RM[R, C]$ is an input signal that is asserted when there is a request for at least one packet transfer from input R to output C. If the scheduler grants a transfer from input R to output C, then the output signal $G[R, C]$ is asserted. The scheduler is composed of two main modules as Fig. 14 demonstrates: the *pattern sequence matrix finite state machine* (*PSM-FSM*) and the *cell matrix* (CM). The *PSM-FSM* is responsible for generating the control signals that apply the diagonals. It has as inputs the TSC and the IC clocks. During each IC cycle, the *PSM-FSM* asserts one signal of the N diagonal signals $D_0, D_1, \ldots, D_{N-1}$. The signals are asserted within each TSC cycle in the order directed by the pattern sequence matrix. The *PSM-FSM* can be easily implemented, since it simply executes Algorithm 1.

The cell matrix is an $N \times N$ matrix of functionally equivalent cells. Each cell $CM[R, C]$ inputs the request $RM[R, C]$ and is responsible for asserting its grant signal, $G[R, C]$, when the corresponding request is granted. The cells in CM operate in parallel, so that the delay of each calculated grant signal is minimized.

Each of the diagonal signals is provided as input to the cells of the cell matrix that are in the given diagonal pattern, e.g., D_0 is input at cells $CM[0, 0]$, $CM[1, 1]$, ..., $CM[N-1, N-1]$. Note that there is exactly one diagonal signal that is input to each cell. This occurs because each entry of the request matrix is covered by exactly one diagonal as is described in Algorithm 1.

The grant signals, G, are calculated (asserted) by the cells of the cell matrix, which employ the design shown in Fig. 15(a). The cell asserts its grant signal when the request signal, $RM[R, C]$, the diagonal signal, D_K ($0 \leq K \leq N-1$), and the signals $AVI[R]$ and $AVO[C]$ are asserted, or if the aforementioned grant signal was asserted in the previous IC cycle and the *Reset* signal has not occurred. The clock of the D-flip-flop in the cell is IC. This flip-flop is reset at the beginning of each TSC cycle. The signals $AVI[R]$ and $AVO[C]$ represent the availability of input R and output C, respectively, for granting a request, i.e., when $AVI[R]$ is asserted, it means that the input R has not been allocated earlier during the TSC cycle and

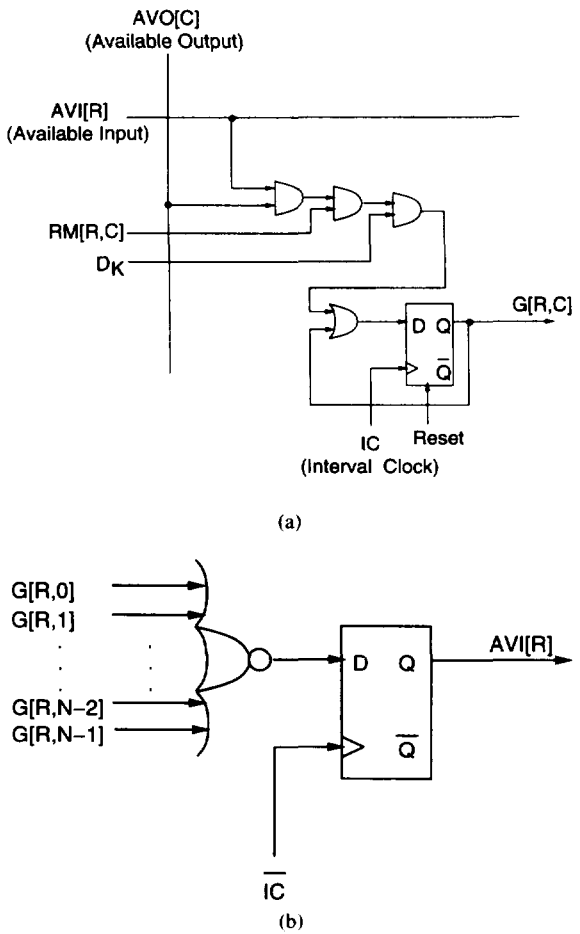


Fig. 15. 2DRR cell design, (a) cell design, (b) AVI design.

is thus available for allocation during the current IC cycle. Similarly, the assertion of the $AVO[C]$ signal indicates that output C has not been allocated earlier. Fig. 15(b) shows the calculation of the $AVI[R]$ signal. $AVO[C]$ is calculated in a similar way. In the above design we have assumed that a requesting input will negate its request signal (i.e., set it to zero) when it is served if no further queued requests exist.

A prototype implementation of a 4×4 Basic 2DRR scheduler that uses the above design has been built. This prototype uses two ALTERA [1] cell arrays, one for the *PSM-FSM* and one for the 4×4 cell matrix. The prototype operates with an IC clock period of 40 ns and thus demonstrates that the Basic 2DRR algorithm is suitable for implementation in high-speed switches. A scaled-up version of this simple prototype could readily support an ATM switch operating at 155 Mbps (SONET OC-3 rate) link speeds for switch sizes that are 64×64. For a 1 Gb/s link speed, the prototype could directly be used for switch sizes of up to 8×8. Further, through the use of a custom or semicustom implementation the IC clock could be significantly sped up and thus the scheduler could support switches of size 16×16 or larger for the 1 Gb/s link speed since the cell computation time varies with $O(N)$.

B. Enhanced 2DRR Scheduler

The architecture of the Enhanced 2DRR scheduler follows a similar approach to that described above, i.e., the grant

signals are calculated in parallel. The information that is needed at each cell is the same as was needed for the Basic 2DRR scheduler and the calculation of the CM cells is the same. The fundamental difference in the case of the Enhanced 2DRR scheduler is that the cells belong to different diagonals in different phases. So, although in the Enhanced 2DRR scheduler architecture we can use the same cell as was used in the Basic 2DRR scheduler for the calculation of $G[R, C]$, there is a difference in the selection process of a cell, i.e., the calculation of D_K. In the Basic 2DRR scheduler, a cell with coordinates R and C is selected by the *PSM-FSM* when the diagonal number that is selected for application in an interval matches the entry $DM[R, C]$ in the diagonal matrix. For the Basic 2DRR scheduler the diagonal $DM[R, C]$ is static (i.e., constant over time) and is given by (1). This is not the case for the Enhanced 2DRR scheduler.

To design the cell selection process for the Enhanced 2DRR scheduler, we need to either move the selection decision to each cell, or to change PM, and consequently the operation of *PSM-FSM* every N cycles. The second alternative is complex and costly as the size of the scheduler increases. Since each diagonal covers different cells during each phase, a centralized scheduler would require N^2 distinct output wires to control all N^2 cells of the cell matrix as well as a memory with N^3 entries. For small size switches, this approach can provide a solution, but when N becomes large, the approach can be quite costly. For such large designs we propose a design that uses the first alternative, i.e., to move the selection decision to the cells. To design this alternative, we need to have every cell with coordinates R and C test in every interval whether the following equality holds:

$$D(I, J) = (C - PM[R, P]) \bmod N, \qquad (15)$$

where $D(I, J)$ is the diagonal number that is broadcasted to all of the cells in the cell matrix during the Ith interval of the Tth TSC system cycle, and where $J = T \bmod N$, and $P = \lfloor (T \bmod N^2)/N \rfloor$ is the phase number. Equation (15) is derived from (14) where the index T indicates that the system cycle is the T-th system cycle since the switch was initialized. With the approach of (15), each cell needs to read matrix PM every N cycles to obtain the entry $PM[R, P]$. Since this information is different for every cell, the memory that stores PM will be a *hot-spot* and thus the involved delays will be high.

To avoid this problem, we distribute the control on a per row basis, as is shown in Fig. 16. There is a controller in each row, the *row controller*. In each interval, the row controller broadcasts the index C of the selected cell to all of the cells in its row. In turn, cells check, in parallel, whether the broadcast index C matches their own column index. The cell of the row that has a match is selected to calculate its grant signal, G, as was done for the Basic 2DRR scheduler (with the same circuitry as was shown in Fig. 15 where the D_K signal is the *coincidence* signal). Each row controller calculates the index C that it broadcasts using the following equation:

$$C = (PM[I, J] + PM[R, P]) \bmod N, \qquad (16)$$

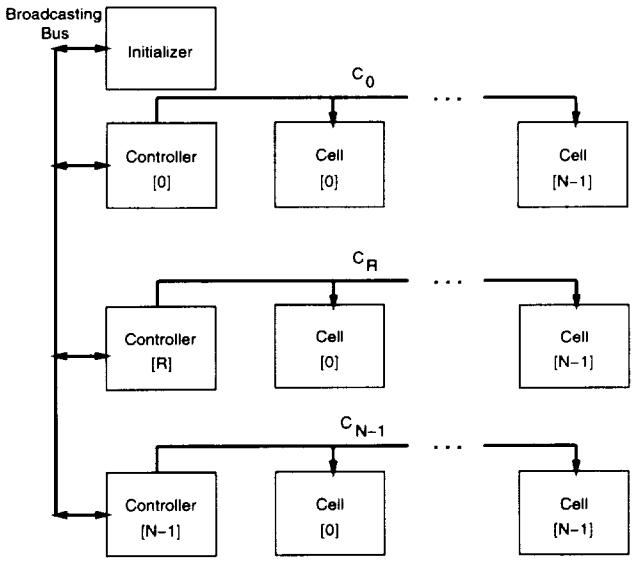


Fig. 16. Enhanced 2DRR design.

where I, J, R, and P are the same as was defined for equation (15).

The Enhanced 2DRR scheduler architecture is shown in Figs. 16 and 17 and operates as follows. During initialization, the *initializer* calculates the matrix PM and stores it in the *row controllers* on a row basis, i.e., $row\ controller[R]$ stores $row[R]$ of PM. Since $row\ controller[R]$ has $row[R]$ of PM locally stored, it can obtain $PM[R, P]$ by directly accessing its local memory. $PM[I, J]$ is the datum needed by all of the *row controllers*, and is stored in the local store of $row\ controller[I]$. For this reason, the operation of the controllers is divided into two phases: *broadcast* and *calculation*. During the *broadcast phase*, only $row\ controller[I]$ operates and broadcasts to all of the remaining controllers the entry $PM[I, J]$ that is stored in its local memory. This is performed over the *broadcasting bus* (see Fig. 17). During the *calculation phase*, each controller accesses its own local memory to obtain the entry $PM[R, P]$, which it then adds to the latched entry $PM[I, J]$ that was broadcasted during the preceding *broadcast phase*. In this fashion, each *row controller* calculates the index C that is active during this interval, as shown in Fig. 17, and broadcasts it to all of the cells in its row. The cell whose column index matches C is activated and calculates the corresponding grant signal, G, in exactly the same way as was described for the Basic 2DRR scheduler.

VIII. Summary and Conclusions

We have described two new types of computationally efficient scheduling algorithms for advanced input queueing switching architectures. These two types of 2DRR schedulers, the Basic and Enhanced 2DRR algorithms, have the same saturation throughput (of one) that is achieved with an output queueing architecture. However, compared with a simple input queueing architecture, the 2DRR schedulers require only a small increase in queue complexity (i.e., advanced input queueing) and the addition of our simple scheduler hard-

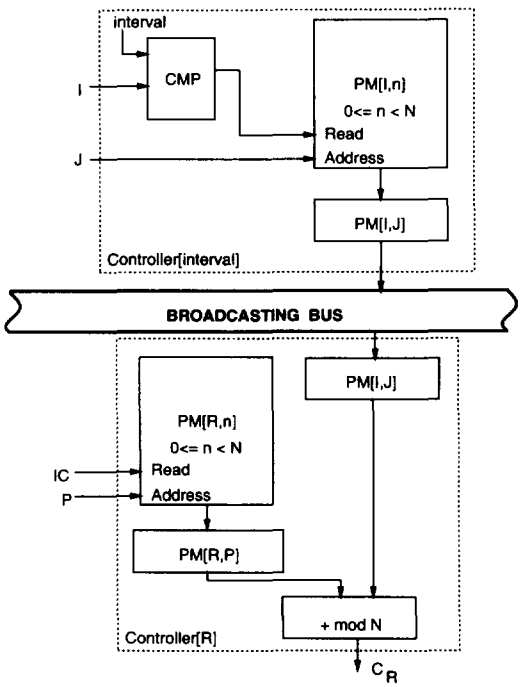


Fig. 17. Row controller organization.

ware to achieve this efficiency, whereas the output queueing architecture requires an N times internal speed-up.

We analyzed the fairness guarantees that each of the 2DRR scheduling algorithms provides and demonstrated the fairness advantages of the Enhanced algorithm for certain traffic patterns. While the Enhanced 2DRR algorithm is fairer than the Basic 2DRR algorithm, it does require some additional implementation complexity.

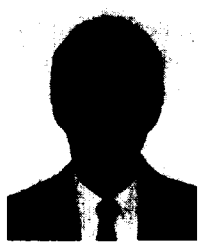
Both of the 2DRR schedulers can be implemented using simple logic components thereby allowing a very high-speed switch implementation. We have designed efficient hardware implementations for both the Basic and Enhanced 2DRR scheduler and have constructed a prototype of the Basic 2DRR scheduler.

ACKNOWLEDGMENT

We gratefully acknowledge T. Pecoraro for his help in the implementation of the 4×4 scheduler prototype.

REFERENCES

[1] *The Maximalist Handbook*, Altera Corporation, 1990.
[2] M. A. Bonuccelli, I. Gopal, and C. K. Wong, "Incremental time-slot assignment in SS/TDMA satellite systems," *IEEE Trans. Commun.*, vol. 39, pp. 1147-1156, 1991.
[3] W. T. Chen, H. J. Liu, and Y. T. Tsay, "High-throughput cell scheduling for broadband switching systems," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1510-1523, Dec. 1991.
[4] I. Gopal, D. Coppersmith, and C. K. Wong, "Minimizing packet waiting time in a multibeam satellite system," *IEEE Trans. Commun.*, vol. 30, pp. 305-316, 1982.
[5] M. G. Hluchyj and M. J. Karol, "Queueing in high-performance packet switching," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1587-1597, Dec. 1988.
[6] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. 27, pp. 1449-1455, 1979.
[7] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Commun.*, vol. COM-35, pp. 1347-1356, 1987.
[8] M. J. Karol, K. Y. Eng, and H. Obara, "Improving the performance of input-queued ATM packet switches," in *Proc. INFOCOM '92*, pp. 110-115, 1992.
[9] L. Kleinrock, *Queueing Systems, Vol. II: Computer Applications*. New York: Wiley, 1976, Section 4.4.
[10] H. Matsunaga, and H. Uematsu, "A 1.5 Gb/s 8x8 cross-connect switch using a time reservation algorithm", *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1308-1317, 1991.
[11] C. Rose, "Rapid optimal scheduling for time-multiplex switches using a cellular automaton," *IEEE Trans. Commun.*, vol. 37, pp. 500-509, 1989.



Richard O. LaMaire (M'87) received the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1987.

He is currently a Research Staff Member at the IBM T. J. Watson Research Center. At M.I.T., he conducted research in the areas of adaptive and digital control theory and estimation. For the past seven years, he has conducted research in the communications area focusing initially on wired local area networks (LAN's) and more recently on wireless LAN's and personal communication networks. His interests include wireless communications systems, scheduling and switching algorithms, and communications system design.

Dr. LaMaire is currently serving as a feature editor for the IEEE PERSONAL COMMUNICATIONS magazine. His email address is: lamaire@watson.ibm.com

Dimitrios N. Serpanos (M'90) for a photograph and biography, please see the August 1994 issue of this TRANSACTIONS, p. 362.