Matching Output Queueing with a Multiple Input/Output-Queued Switch

Hyoung-Il Lee
Network Research Laboratory, ETRI
161 Gajeong-dong, Yuseong-gu, Daejeon 305-350, Korea
Email: hil@etri.re.kr

Seoung-Woo Seo
School of Electrical Engineering and Computer Science
Seoul National University, Seoul, Korea 151-742
Email: sseo@snu.ac.kr

***Abstract*— In this paper, we show that the *multiple input/output-queued (MIOQ)* switch proposed in our previous paper [22] can emulate an output-queued switch only with two parallel switches. The MIOQ switch requires no speedup and provides an exact emulation of an output-queued switch with a broad class of service scheduling algorithms including FIFO, weighted fair queueing (WFQ) and strict priority queueing regardless of incoming traffic pattern and switch size. First, we show that an $N \times N$ MIOQ switch with a $(2, 2)$-*dimensional crossbar fabric* can exactly emulate an $N \times N$ output-queued switch. For this purpose, we propose the *stable strategic alliance (SSA)* algorithm that can produce a stable many-to-many assignment, and then apply it to the scheduling of an MIOQ switch. Next, we prove that a $(2, 2)$-dimensional crossbar fabric can be implemented by two $N \times N$ crossbar switches in parallel for an $N \times N$ MIOQ switch. For a proper operation of two crossbar switches in parallel, each input-output pair matched by the SSA algorithm must be mapped to one of two crossbar switches. For this mapping, we propose a simple algorithm that requires at most $2N$ steps for all matched input-output pairs. In addition, to relieve the implementation burden of N input buffers being accessed simultaneously, we propose a buffering scheme called *redundant buffering* which requires two memory devices instead of N physically-separate memories.**

I. Introduction

The high bandwidth support of recent switches and routers makes traditional output-queued or shared-memory architectures inappropriate due to their strict requirement that output memories have to be operated at a rate at least equal to the aggregate of all the input links. Therefore, a number of high-speed switches and routers [1]-[2] employ an input-queued architecture where memories and switch fabric may operate at only the line speed. However, although many scheduling algorithms [3]-[7] proposed for the input-queued architecture achieve 100% asymptotic throughput, none of these algorithms match the performance of an output-queued switch. Also, when a switch is required to provide QoS guarantees, the output-queued architecture is more convenient than the input-queued architecture because most of the schemes developed for providing QoS guarantees were based on the output-queued architecture. In these schemes, one of service scheduling algorithms [8]-[11] is placed at output ports and used to control the latency of packets which are immediately placed in output buffers upon arrival at input ports.

The speedup of the switch fabric has been proposed as one solution to improve the performance of an input-queued switch. The speedup of s enables a switch to remove up to s packets from each input and delivers up to s packets to each output within a time slot, where a time slot is the time between packet arrivals at input ports. Since this speedup requires buffering at outputs as well as inputs, this architecture is called a *combined input and output queued(CIOQ)* switch.

Then, a natural question about a CIOQ switch arises, i.e., how much speedup is sufficient to match the performance of an output-queued switch? There have been many research works on this subject [12]-[17]. In [15], the authors showed that a CIOQ switch with VOQs operating under the *lowest occupancy output first algorithm (LOOFA)* and speedup of 2, is work-conserving for all traffic patterns and switch sizes. In [16], it was shown that a wide class of *maximal size matching algorithms* (including well-known scheduling algorithms such as iSLIP [3][4] and 2DRR [6]) can provide the same throughput performance of an output-queued switch with speedup equal to 2. Moreover, in [17], a speedup of 2 in a CIOQ switch was proven to be sufficient to emulate an output-queued switch exactly using the *stable marriage matching (SMM)* algorithm which is well-known by the name of *Gale-Shapley (GS) algorithm* [18].

However, as remarked in [21], the speedup of the switch fabric needs memories with shortened access time and requires a scheduler to make scheduling decisions within reduced time, which is quite difficult even for a speedup of only 2. Therefore, as a recent approach to achieve the comparable performance of an output-queued switch without any speedup, a *parallel switching architecture(PSA)* has been studied [19]-[21] which is composed of input demultiplexers, output multiplexers and parallel switches. In a PSA, an incoming stream of packets is spread packet-by-packet by a demultiplexer across the parallel switches, then recombined by a multiplexer at each output.

In [19], it is shown that a PSA with k parallel switches can emulate an output-queued switch with any QoS queueing discipline if each parallel switch operates at a rate of $3R/k$, where R is a link rate. However, since each parallel switch in the proposed scheme is assumed to be an output-queued switch and the emulation of output queueing by a CIOQ switch requires a speedup of 2, its finally-required operation rate is $6R/k$. Therefore, to achieve the same performance of an output-queued switch without any speedup, the proposed architecture needs 6 parallel switches. In [20], the authors adopt

the *co-ordination buffers* instead of a complex centralized distribution algorithm used in [19] to avoid the mis-sequence problem among cells from different parallel switches. The conclusion is that the proposed PSA with no speedup can mimic a FIFO (First-In First-Out) output-queued switch with a relative queueing delay bound of $2N$ time slots. As another solution to guarantee the FIFO sequence of cells in a PSA, the authors in [21] propose a scheme of applying the *same matching* at each of the parallel crossbar switches during each slot with an appropriate demultiplexing strategy at the inputs. However, in [21], no necessary conditions are given to emulate output queueing.

In this paper, we show that a modified version of the *multiple input/output-queued(MIOQ)* switch proposed in our previous paper [22] can emulate an output-queued switch only with two parallel switches. (See also Section II for brief review of the MIOQ switch.) The modified MIOQ switch is equipped with parallel switches instead of a (k, m)-*dimensional crossbar fabric* in the original one. First, we show that a $(2, 2)$-dimensional crossbar fabric which is equivalent to a $2N \times 2N$ crossbar switch is sufficient for an $N \times N$ MIOQ switch to emulate an $N \times N$ output-queued switch exactly without any speedup. In other words, we can make an $N \times N$ output queued switch with a $2N \times 2N$ crossbar switch operated at only the line rate. For this purpose, we propose an algorithm for the *stable strategic alliance problem* defined newly in this paper and prove its finite, stable and deterministic properties. The proposed algorithm which we call the *stable strategic alliance(SSA)* algorithm has the same number of iterations as the SMM algorithm used for a CIOQ switch to emulate an output-queued switch with a speedup of two. It is noted that the emulation of output queueing with an MIOQ switch holds for all incoming traffic patterns, any size of switches, and a broad class of service scheduling algorithms such as FIFO, WFQ and strict priority queueing.

Then, we prove that a $(2, 2)$-dimensional crossbar can be replaced by two $N \times N$ crossbar switches in parallel to achieve the same result. To resolve possible conflicts in a crossbar switch, we need a mapping of each input-output pair matched by the SSA algorithm to one of the two crossbar switches. For this mapping, a simple algorithm and its hardware implementation are also proposed. In addition, to relieve the implementation burden for N input buffers to be accessed simultaneously, we propose a buffering scheme called *redundant buffering* which requires two physical memories instead of N physically-separate memories. In this scheme, two of N input buffers can be accessed simultaneously, which is a necessary condition for the MIOQ switch to emulate an output-queued switch. Based on all these proofs, we finally conclude that two crossbar switches in parallel and two physical memories at each input are sufficient for an MIOQ switch to mimic an output-queued switch exactly with no speedup.

The rest of this paper is organized as follows. Section 2 describes the general architecture of an MIOQ switch. Section 3 defines the stable strategic alliance problem formally, and proposes an algorithm for this problem. In section 4, we prove the sufficiency of a $(2, 2)$-dimensional crossbar for the emulation of output queueing with an MIOQ switch. In Section 5, we show that two crossbar switches can also be used for the output queueing emulation. In Section 6, we describe a buffering scheme to implement N input buffers with two physical memories. Finally, concluding remarks are presented in Section 7.

II. MIOQ Switch Architecture

In this section, we briefly review the MIOQ switch architecture for convenience' sake and introduce a modified version of the MIOQ switch composed of parallel crossbar switches. Basically, an MIOQ switch architecture of size N is composed of N input buffers at each input, M output buffers at each output, a (k, m)-dimensional crossbar fabric and a scheduler. Figure 1 illustrates an $N \times N$ MIOQ switch with a $(3, 3)$-dimensional crossbar fabric.

In this architecture, N input buffers at each input are used for *virtual output queueing* to eliminate a well-known *head-of-line blocking*. A (k, m)-dimensional crossbar fabric, which we abbreviate to a (k, m)-*dim crossbar*, provides k ingress lines for each input and m egress lines for each output. So, in a (k, m)-dim crossbar, up to k cells can leave from each input and up to m cells can be delivered to each output within a time slot. It is noted that, since no speedup is assumed in an MIOQ switch, only one cell from each input buffer can be selected and moved to one of k ingress lines via an $N \times k$ interconnection network.

To exclude the necessity of speedup at outputs, we divide an output buffer into multiple ones, each of which is accessed only once in a time slot. For a FIFO scheduling discipline, we need m separate buffers at each output with a queue management scheme to avoid mis-sequencing of cells, which is based on a round-robin rule. The queue management scheme inserts at most m cells arriving at an output from m egress lines to m buffers in a round-robin order and takes out cells one-by-one according to the same round-robin order. The basic operation of this scheme is the same as the shifter used to balance the traffic loads among multiple output buffers in the Knockout switch [23]. We adopt this scheme in the MIOQ switch for guaranteeing the order of cells which depart from the same input. Figure 2 illustrates the queue management scheme for three output buffers at an output.

For other scheduling disciplines such as WFQ and strict priority queueing, we need N physically-separated buffers at each output, each of which corresponds to an input, and in each buffer, a memory space is logically allocated for each session or class. Since more than two cells from an input are not delivered to an output simultaneously in a time slot (as described above, only one cell is allowed to be removed from an input buffer at each input within a time slot), at most m cells from egress lines at an output can be moved to corresponding output buffers according to their inputs respectively with no speedup.

The scheduler in an MIOQ switch makes a scheduling decision in every time slot that includes which input buffers to

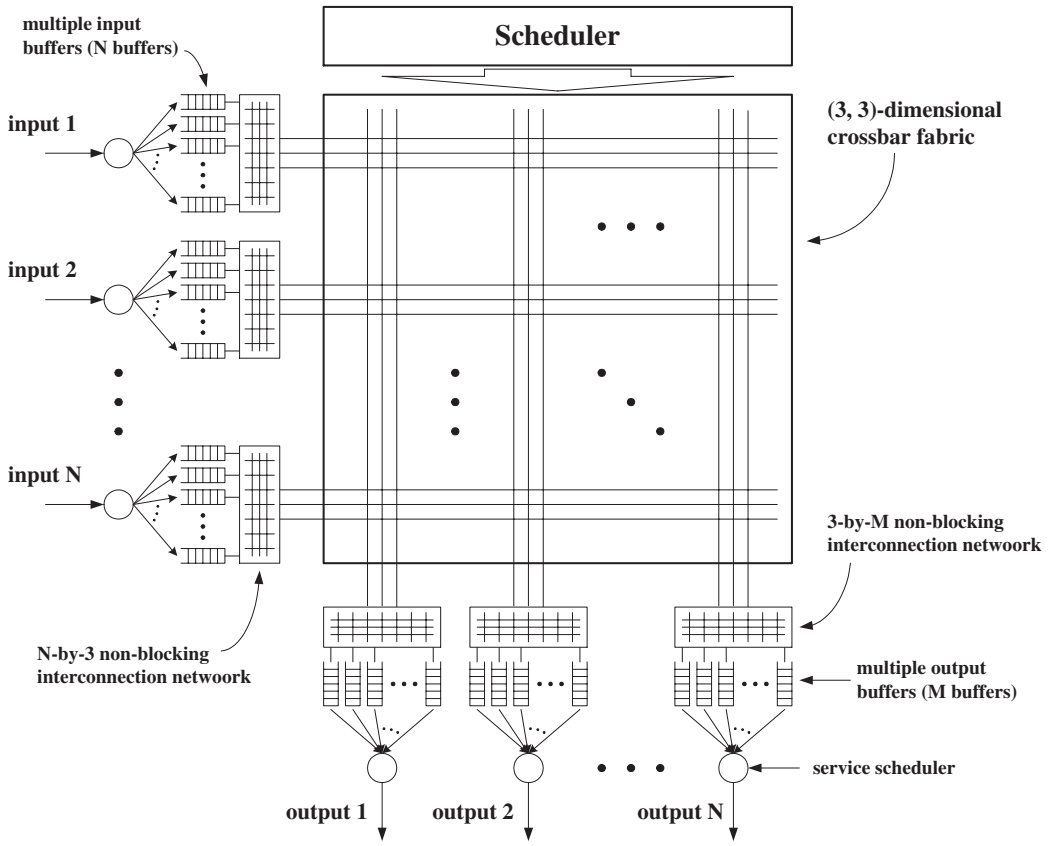


Fig. 1. An $N \times N$ MIOQ switch architecture with (3, 3)-dimensional crossbar fabric

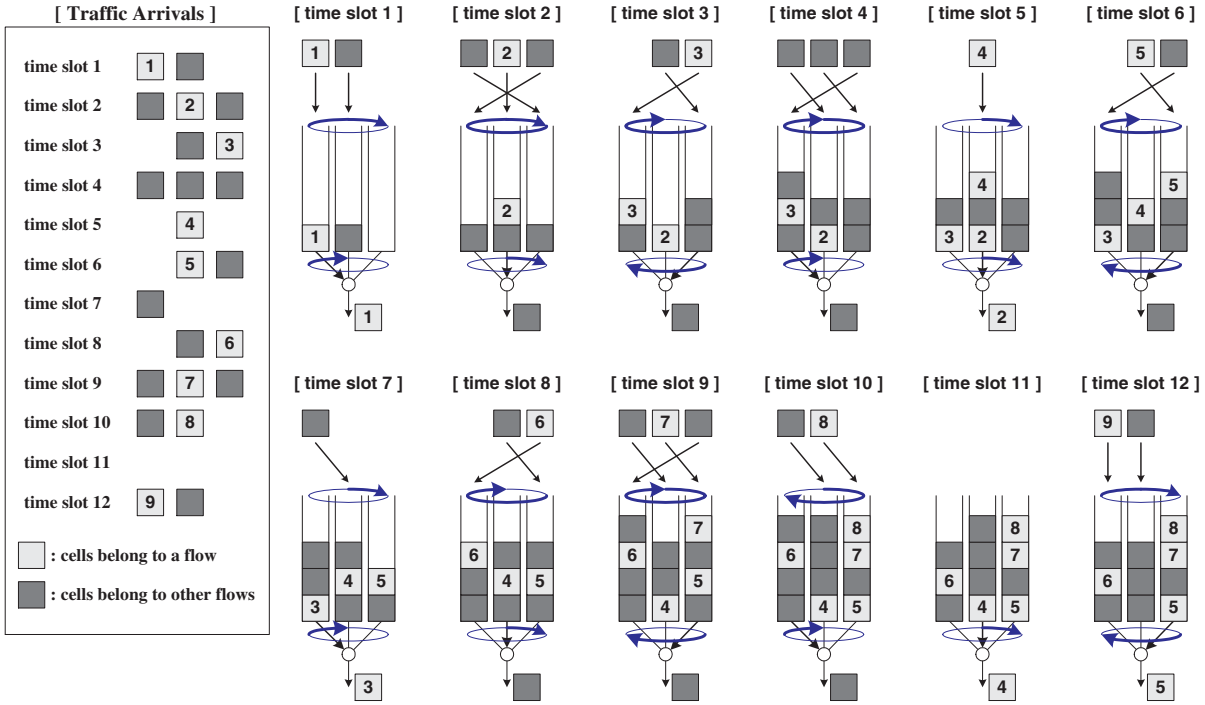


Fig. 2. A simple round-robin-based queue management scheme for FIFO output queueing

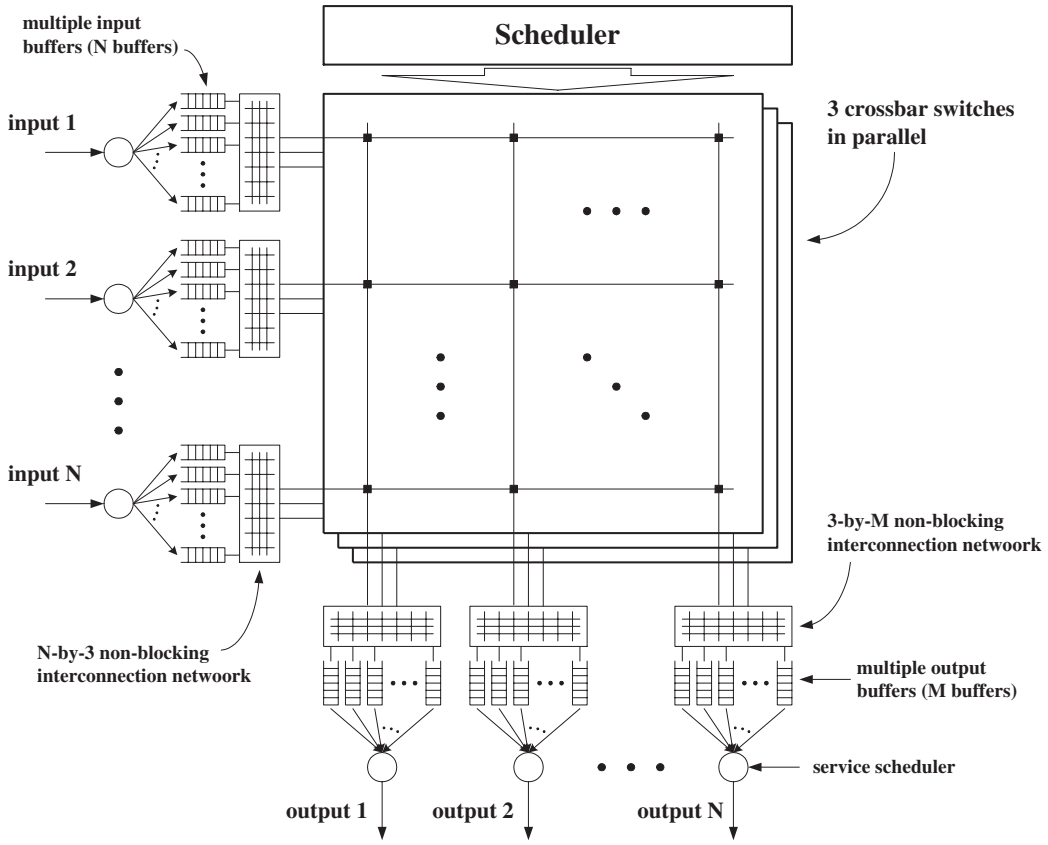


Fig. 3. An $N \times N$ MIOQ switch architecture with 3 crossbar switches in parallel

be read, which output buffers to be written and a configuration of the (k, m)-dim crossbar. In the first scheduling step, each input sends a request message to the scheduler which includes the information of the head-of-line cell in each input buffer. Since only one cell can departs from an input buffer in a time slot, cells after the head-of-line one need not be considered in a scheduling process. Then, the scheduler selects cells to be transferred in a time slot among at most N^2 cells and configures the (k, m)-dim crossbar as well as interconnection networks.

Also, an MIOQ switch can be converted into a parallel switching architecture using $max\{k, m\}$ crossbar switches in parallel instead of a (k, m)-dim crossbar, which will be shown in Section 5. As shown in Figure 3 depicting the modified MIOQ switch with 3 crossbar switches in parallel, each of ingress and egress lines is connected to one of parallel switches. When multiple cells from an input are transmitted to different outputs in this architecture, each cell have to be allocated to one of parallel switches. Also, more than two cells destined to an output is not allocated to a parallel switch, which avoids contention for an output.

III. Stable Strategic Alliance Algorithm

The scheduling problem for an MIOQ switch to emulate output queueing forms a novel *stable assignment problem* with conditions different from traditional problems such as *stable marriage problem*, *stable roommate problem* and *stable residents/hospitals problem* [24]. The goal of a stable assignment problem is to find a *bipartite graph* between two sets which satisfies a given stability constraint when a preference list of elements in the opposite set is given for each element.

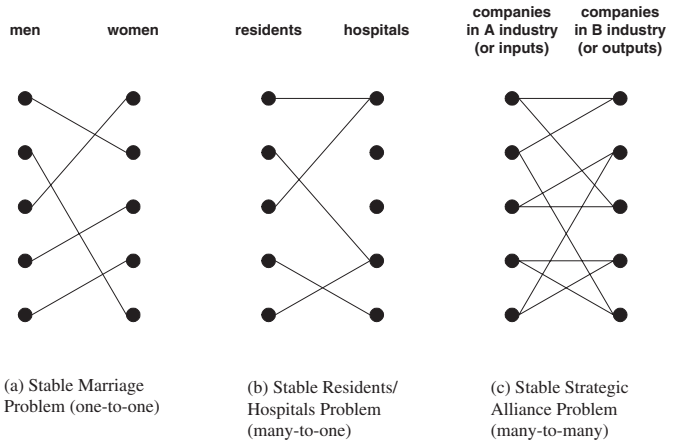


Fig. 4. Stable assignment problems between two sets

As illustrated in Figure 4 (a), the goal of the stable marriage problem is to find a one-to-one assignment or a

matching which satisfies the following stability condition:

Definition 1: *A marriage matching is stable if there is no man-woman pair for the matching who are mutually acceptable, where*

- *he prefers her to his assigned woman;*
- *and simultaneously, she prefers him to her assigned man.*

Because of its one-to-one assignment property, this stable marriage matching has been applied to scheduling for an input-queued switch as in [17]. The stable residents/hospitals problem can be thought of as a marriage problem, in which polygamy is allowed by the hospitals, and covers the situations to find a stable many-to-one assignment between two sets.

On the other hand, a scheduling for an MIOQ switch is a many-to-many assignment problem, in which polygamy is allowed by both sides of inputs and outputs. Therefore, we newly define a stable many-to-many assignment problem, named *stable strategic alliance(SSA) problem* from frequent many-to-many alliances between companies, as follows:

Definition 2: *The goal of the stable strategic alliance problem is to find a stable many-to-many assignment with given conditions:*

- *two sets of the same size N: set A and B;*
- *a preference list of elements which belong to the opposite set for each element;*
- *an element of set A can be matched with up to k elements of set B, and an element of set B can be matched with up to m elements of set A.*

The stability condition of the strategic alliance problem is given as follows:

Definition 3: *A many-to-many assignment is stable provided that there is no element α of set A and element β of set B with the properties that*

- *α is not fully matched (matched with less than k elements), or would prefer β to at least one of elements which has been matched with;*
- *and simultaneously, β is not fully matched (matched with less than m elements), or would prefer α to at least one of elements which has been matched with.*

Then, we propose an algorithm for the problem defined above, which we call a *stable strategic alliance(SSA) algorithm*. The basic idea of the SSA algorithm stems from the *Gale-Shapley(GS) algorithm* [18] which is a well-known solution for the stable marriage problem. The SSA algorithm proceeds by rounds each of which is composed of two parts: (1) elements of one set (called *active set*) make proposals of matching; (2) elements of the other set (called *passive set*) reject or (tentatively) accept the proposals. Reversing their roles of the two sets, the SSA algorithm produces a vastly different assignment which is also a stable solution for the SSA problem.

In the first part, elements of the active set which are currently fully matched do nothing. Each element which is not fully matched makes a new proposal to the highest one in its preference list which has not already rejected it, whether or not the element to be proposed is fully matched. In the second

```
inputs : preference lists for each element
outputs : matching lists for each element

k = max_of_matched_of_set_A;
m = max_of_matched_of_set_B;
for each element a of set A
    #_of_matched(a) = 0;
    #_of_remained(a) = number of elements in the preference list of a;
endfor
for each element b of set B
    #_of_matched(b) = 0;
endfor
while there is element a of set A for #_of_matched(a) < k and #_of_remained(a) > 0
    for each element a of set A
        if #_of_matched(a) < k and #_of_remained(a) > 0, then {
            b = the highest one on a's preference list;
            add a to b's proposal list;
            remove b from a's preference list;
            #_of_remained(a) = #_of_remained(a) - 1; }
    endfor
    for each element b of set B
        while b's proposal list is not empty
            a = the highest one on b's proposal list;
            c = the lowest one on b's matching list;
            if #_of_matched(b) < m, then {
                add a to b's matching list;
                add b to a's matching list;
                #_of_matched(b) = #_of_matched(b) + 1;
                #_of_matched(a) = #_of_matched(a) + 1; }
            else if preference of a < preference of c, then {
                remove b from a's matching list;
                remove a from b's matching list;
                add c to b's matching list;
                add b to c's matching list;
                #_of_matched(a) = #_of_matched(a) - 1;
                #_of_matched(c) = #_of_matched(c) + 1; }
            else, then{
                break while; }
        end while
    endfor
end while
```

Fig. 5. A pseudo code of the stable strategic alliance algorithm

part, an element of the passive set which is not fully matched accepts the given proposals until fully matched. After then, the element first selects the proposal with the highest preference among the given proposals which we call the *most attractive proposal (MAP)*, and compares its preference with that of the lowest-preferred one among already-matched elements called the *reluctantly-allied company (RAC)*. If the element prefers the MAP to the RAC, it breaks the current alliance with the RAC and accepts the MAP. Otherwise, the element rejects all current proposals including the MAP. In the former case, the algorithm repeats the above procedure until no proposal remains. A pseudo code of the SSA algorithm is given in Figure 5.

The specifications of the SSA algorithm made so far are not completely satisfactory as a solution for our problem. To be a complete solution, the algorithm must provide answers for three questions as follows:

1. Does the proposed SSA algorithm terminate finitely?
2. Is a many-to-many assignment produced by the SSA algorithm stable?
3. Does the SSA algorithm produce the deterministic result regardless of the order of proposals and responses? (This question needs to be proved because the description of the SSA algorithm does not specify the order in which elements of the active set make proposals and the order in which elements of the passive set respond.)

In other words, we must show its finite, stable and deterministic properties as follows:

Theorem 1: *The SSA algorithm terminates finitely.*

Proof: No element of the active set proposes to the same element of the passive set twice, so each element of the active set can make at most N proposals. Altogether, all elements of the active set can make no more than N^2 proposals. In each round, at least one proposal is made; so there can be at most N^2 rounds. $\diamond$

Theorem 2: *An assignment produced by the SSA algorithm satisfies the stability condition in Definition 3.*

Proof: We consider two cases:

(1) If an arbitrary element α of the active set is not fully matched, it must have proposed to all unmatched candidates in its preference list before the algorithm termination, which is the termination condition of the SSA algorithm. If one of the candidates is not fully matched or prefers α to its RAC, it has accepted α's proposal in some round and is finally matched up with α. Therefore, all candidates unmatched with α are fully matched and prefer their RACs to α.

(2) If an arbitrary element α of the active set prefers another element β unmatched with α to α's RAC, α must have proposed to β in some round according to the SSA algorithm. If β is not fully matched or prefers α to β's RAC, β has accepted α's proposal in that round and is finally matched up with α. Therefore, β is fully matched and prefers its RAC to α. $\diamond$

Theorem 3: *The assignments produced by the SSA algorithm are the same, no matter how we determine the order of proposals and responses.*

Proof: In a round, no proposal of any element is affected in any way by what another element of the active set does. Likewise, no element's response to its proposals is affected by what other elements of the passive set do. So, in each round, it does not matter in what order the proposals are made or in what order the elements proposed make their choices. $\diamond$

IV. Emulation of an Output-Queued Switch

In this section, we show that if an MIOQ switch with $(2, 2)$-dim crossbar fabric is scheduled by the SSA algorithm, it can exactly mimic a shadow output-queued switch that is assumed to be fed with the same input traffic pattern as an MIOQ switch.

A. Definitions

In the rest of this paper, we use the following terms originally defined in [17], and revisit them in this section before proceeding.

Definition 4: *A "push-in first-out (PIFO) queue" is a queue in which arriving cells are placed at arbitrary location, and the cell at the head of queue is always the next to depart.*

Definition 5: *The "time to leave" for cell c [TL(c)] is the time slot at which c will leave the shadow output-queued switch.*

Definition 6: *At any time, the "output cushion" of a cell c [OC(c)] is the number of cells waiting in the output buffer at cell c's output port with a smaller time to leave value than cell c.*

Definition 7: *At any time, the "input thread" of a cell c [IT(c)] is the number of cells ahead of cell c in its priority list of its corresponding input.*

Definition 8: *At any time, the "slackness" of a cell c [L(c)] equals the output cushion of cell c minus its input thread, i.e., L(c) = OC(c) - IT(c).*

B. Operations of an MIOQ switch for Output Queueing Emulation

For the emulation of output queueing, we consider an MIOQ switch with a $(2, 2)$-dim crossbar in which all input and output buffers are assumed to follow the PIFO queueing rule in Definition 4. Each input has a single priority list and N input buffers each of which corresponds to each output, while each output also has a single priority list and N output buffers each of which corresponds to each input. Let $IQ(i, j)$ and $OQ(i, j)$ denote the input buffer at input i for cells destined for output j and the output buffer at output j for cells departing from input i, respectively.

To describe the operations of the MIOQ switch, we will divide each time slot into three phases:

(1) Arrival phase: New cells arrive at the input ports during this phase.

(2) Scheduling phase: The SSA algorithm selects cells to be transferred from inputs to outputs, and then delivers them via a $(2, 2)$-dim crossbar.

(3) Departure phase: Cells depart from the output ports during this phase.

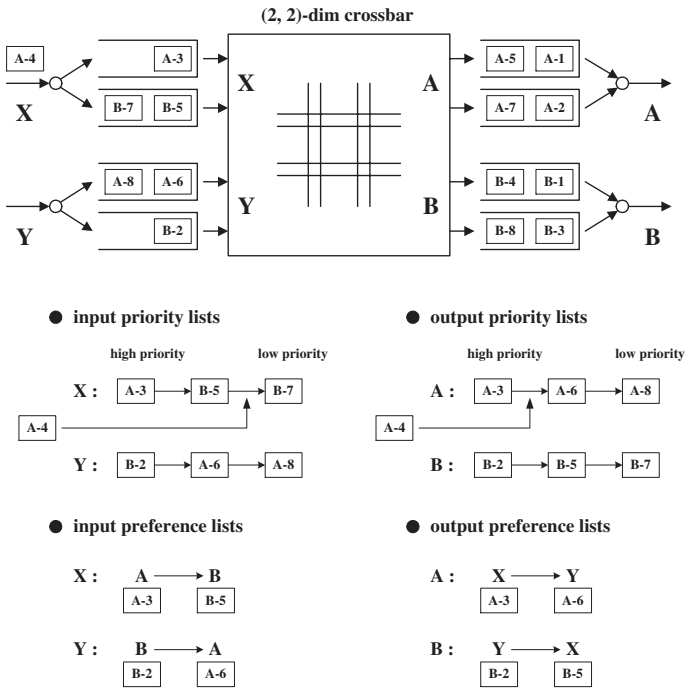


Fig. 6. An example of operation of an 2×2 MIOQ switch with a $(2, 2)$-dim crossbar for output queueing emulation in a time slot

It is noted that an MIOQ switch has only one scheduling phase in a time slot because no speedup is allowed. On the contrary, a CIOQ switch has more than one scheduling phases according to its speedup.

In the arrival phase, when a cell c whose output cushion is u arrives at an input, it is registered into the $(u+1)$th location in the priority list of the input if the list has more than u elements. (In this case, the initial slackness of cell c is zero.) Otherwise, it is registered into the last location of the list. (In this case, cell c has a positive value of its initial slackness.) Also, it is registered to the priority list of its output, where its location is determined by its time-to-leave value. In other words, the cell c is located right prior to the cell with the smallest time-to-leave value that is larger than that of cell c in the priority list. Then, the cell c is inserted into one of input buffers corresponding to its output, where its location is determined according to the order in its input priority list. Hence, cells of an input buffer are located in a sorted manner according to the order in the priority list of the input, and thus, the head-of-line cell of each input buffer has the top priority among the cells in the corresponding input buffer.

In the scheduling phase, we use the SSA algorithm to find a stable assignment between inputs and outputs. To apply the SSA algorithm to an MIOQ switch, first of all, two sets and a preference list for each element remarked in Definition 2 have to be specified. We let two sets contain all inputs and all outputs respectively. A preference list for an input s is given as an ordered list of the head-of-line cells in the input buffers which belong to s according to the order of its priority list. A preference list for an output r is given as an ordered list of the head-of-line cells in the input buffers which are destined for r according to the order in its priority list. When the (input s, output r) pair is selected by the SSA algorithm, the head-of-line cell of $IQ(s, r)$ is transferred to output r and is inserted into $OQ(s, r)$ where its location is determined in a sorted manner according to its time-to-leave value.

Figure 6 shows a snapshot of a 2×2 MIOQ switch with a $(2, 2)$-dim crossbar at the arrival of a cell denoted as A-4, where A represents its output port and 4 represents its time to leave. Since its output A has two cells of which time-to-leave values are lower than 4, its output cushion is 2 so that it is located in the 3th position of the input priority list of X. According to its time-to-leave value, its location in the output priority list of A is determined as the second position. As shown in Figure 6, preference lists for scheduling are also given as an ordered list of the head-of-line cells.

On the output side, the MIOQ switch keeps track of the time to leave for each head-of-line cell of output buffers, and sends out cells at their time to leave during the departure phase. Since cells of an output buffer are located in a sorted manner according to their time-to-leave values, consideration of the head-of-line cells of output buffers is sufficient for output queueing emulation. Therefore, if each cell can be transferred to the output side before its time to leave, the MIOQ switch can mimic output queueing successfully.

C. Sufficiency of a (2, 2)-dim Crossbar

Lemma 1: *If the slackness of a cell in a switch is always nonnegative, the switch exactly mimics an output queued switch that adheres to a PIFO queueing policy, regardless of*

incoming traffic pattern.

Proof: The proof of Lemma 1 is almost identical to that of Theorem 4 in [17], and is omitted here for brevity. ◇

Lemma 2: *In an MIOQ switch with a $(2,2)$-dim crossbar scheduled by the SSA algorithm, the slackness L of a cell c waiting in the input side does not decrease during a time slot.*

Proof: We consider a cell c waiting in the input side. During the arrival phase, the input thread of c can increase by at most one if a newly arriving cell is inserted ahead of c in its input priority list. During the departure phase, the output cushion of c decreases by one. Hence, the slackness of c can decrease by at most 2 during these two phases.

We consider all possible cases occurring during the scheduling phase as follows:

(1) If c is scheduled in this phase, then it is delivered to its output, and we no longer need to be concerned with c.

(2) If the head-of-line cell of c is scheduled in this phase, it decreases its input tread by one and increases its output cushion by one. Therefore, in this case, the slackness is increased by at least 2.

(3) If c itself and its head-of-line cell are not scheduled in this phase, one of following conditions must hold.

a) Two cells that are ahead of c in its input priority list are scheduled in this phase.

b) Two cells that are ahead of c in its output priority list are scheduled in this phase.

When only no or one cell that is ahead of c in its input priority list is scheduled in a scheduling phase, the stability property of the SSA algorithm guarantees that two cells that are ahead of c in its output priority list are scheduled in this phase, which has already proven in Theorem 2. Hence, in this case, the slackness is increased by at least 2.

Therefore, in all cases, the slackness is increased by at least 2 during the scheduling phase and we conclude that the slackness of cell c cannot decrease from time slot to time slot. ◇

Lemma 3: *The slackness of a cell in an MIOQ switch with a $(2,2)$-dim crossbar scheduled by the SSA algorithm is always nonnegative.*

Proof: Because the slackness of an arriving cell is non-negative, the proof naturally follows from Lemma 2. ◇

Theorem 4: *An MIOQ switch with a $(2,2)$-dim crossbar scheduled by the SSA algorithm exactly mimics an output queued switch that adheres to a PIFO queueing policy, regardless of the incoming traffic pattern.*

Proof: The proof follows from Lemma 1 and Lemma 3. ◇

It is noted that, as remarked in [17], the PIFO queueing discipline includes a broad class of scheduling algorithms such as FIFO, WFQ, and strict priority queueing.

V. Mapping to Crossbar Switches in Parallel

To implement an $N \times N$ MIOQ switch for output queueing emulation, we can use a $2N \times 2N$ crossbar switch as a $(2,2)$-dim crossbar. In fact, the two architectures are the same. However, when we have only $N \times N$ crossbar switches, how can we make an $N \times N$ MIOQ switch for output queueing emulation? To answer this question, in this section, we show that a $(2,2)$-dim crossbar can be replaced by two crossbar switches in parallel for an $N \times N$ MIOQ switch to emulate an $N \times N$ output queued switch. To resolve possible conflicts in a crossbar switch, we need a mapping of input-output pairs matched by the SSA algorithm to one of the two crossbar switches.

As the first step, we convert this mapping problem into a well-known *graph coloring problem* [25] by defining a graph G in which nodes denote input-output pairs matched by the SSA algorithm for a $(2,2)$-dim crossbar and an edge between two nodes indicates that they share the same input or output. Then, we show that the given graph G is 2-colorable.

Corollary 1: *According to the classical graph theory [25], if a graph G has no cycles of odd length, G is 2-colorable.*

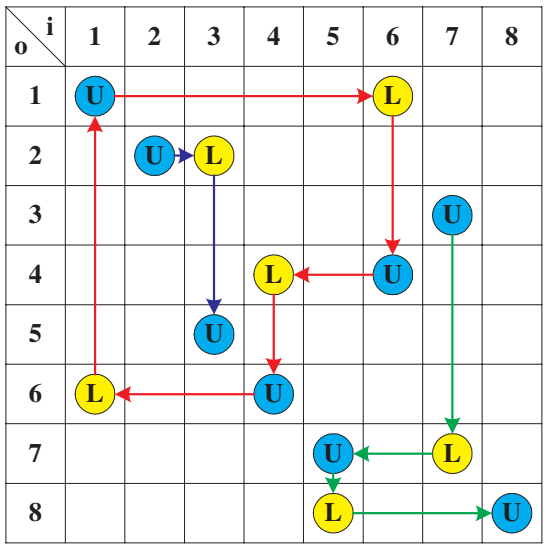
Lemma 4: *The given graph G is 2-colorable.*

Proof: Let us assume that the given graph G has a cycle. Since a node in G has two edges which denote an input and an output respectively, a cycle composed of these nodes has the same number of edges for inputs and outputs. So, if a cycle has k edges for inputs, the total number of edges in the cycle is $2k$. Therefore, if G has a cycle, the length of the cycle is always even. In other words, the given graph G has no cycle of odd length. Therefore, from Corollary 1, G is 2-colorable. ◇

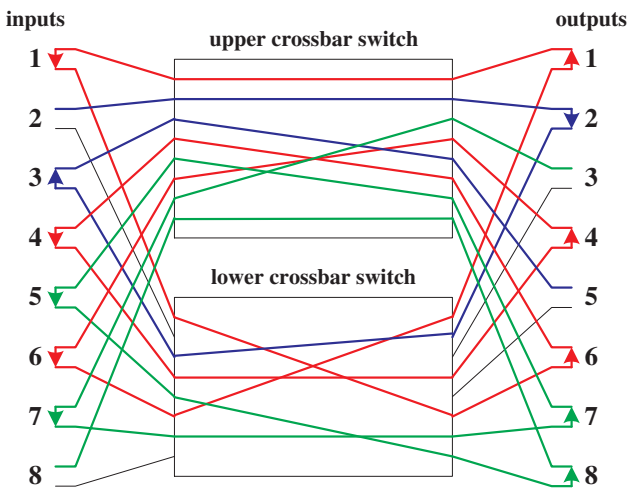
Theorem 5: *Each input-output pair matched by the SSA algorithm for the emulation of an $N \times N$ output queued switch can be mapped to one of two $N \times N$ crossbar switches in parallel with no conflict.*

Proof: In the graph G, let a color represent a crossbar switch. Then, since the given graph coloring problem is equivalent to the mapping problem of matched input-output pairs to one of two crossbar switches, it is directly proven from Lemma 4. ◇

Therefore, we can make an $N \times N$ MIOQ switch with two $N \times N$ crossbar switches in parallel instead of a $(2,2)$-dim



(a) an example of the proposed mapping algorithm for two 8 x 8 crossbar switches in parallel



(b) an example of mapping the results produced by the mapping algorithm to two 8 x 8 crossabar swithses in parallel

Fig. 7. An algorithm for mapping a scheduling result to two crossbar switches in parallel

crossbar, if there is an appropriate scheme for this mapping problem.

Since an $N \times N$ MIOQ switch with n crossbar switches in parallel is composed of three stage switches: $N \times n$ non-blocking interconnection networks, n crossbar switches and $n \times M$ non-blocking interconnection networks, it can be considered as a well-known *Clos network* [26]. According to the *Slepian-Duguid Theorem* [26], for the Clos network to be rearrangeable, it must have at least max(N, M) crossbar switches in the second stage. However, by the SSA algorithm for a $(2, 2)$-dim crossbar, at most two of N inputs in a first-stage switch send packets to two crossbar switches in the second stage and at most two packets arrives at a third-stage switch. Therefore, an MIOQ switch is rearrangeable for the input-output pairs matched by the SSA algorithm if it has at least two crossbar switches in the second stage. (This is the same result derived from the graph coloring problem in Theorem 5.) Therefore, for mapping the input-output pairs matched by the SSA algorithm to one of two crossbar switches, we can use the *Paull's Theorem* [26] repeatedly by inserting the matched input-output pairs sequentially into the rearrangement process. Initially, a matched input-output pair is mapped to one of two crossbar switches. Next, another input-output pair which is matched but not mapped is selected and mapped to one of two crossbar switches through the rearrangement process in the Paull's Theorem. Until all of the matched input-output pairs are mapped, this rearrangement process is repeatedly applied. Note that this mapping scheme requires a sequence of at most $2N$ rearrangement processes each of which needs at most N-1 steps as proven in the Paull's Theorem.

To reduce the computational complexity for the mapping of the matched input-output pairs to one of two crossbar switches, we propose a simple *mapping algorithm* which needs at most $2N$ steps to produce a solution. Initially, the mapping algorithm selects an input-output pair p_1 and maps it to one of two crossbar switches. Next, if there is another input-output pair p_2 which shares the same input or output of p_1 and has not been chosen before, the algorithm assigns the other crossbar switch for p_2, and proceeds with p_2 until all pairs matched by the SSA algorithm are completely visited. Otherwise, the algorithm selects another input-output pair which has not been chosen previously as a new starting point, and proceeds with it until all pairs matched by the SSA algorithm are completely visited. Figure 7 (a) is a visual representation of the mapping algorithm drawn by a chart and illustrates the operation of the proposed algorithm for two 8×8 crossbar switches in parallel. Figure 6 (b) shows the result mapped to the two crossbar switches in parallel.

As a way for hardware implementation of the mapping algorithm, we also propose an arbitration logic called a *crossbar mapper*. As shown in Figure 8, the proposed logic is composed of N^2 switching elements each of which corresponds to each input-output pair. In the proposed logic, two kinds of signals are used, i.e., 1 and -1. If an input-output pair is matched by the SSA algorithm, the corresponding switching element, called *matched switching element (MSE)*, is set in a bar state and inverts its input signal, i.e., from 1 to -1 or from -1 to 1. Otherwise, the corresponding switching element is set in a cross state and passes its input signal transparently. Then, if an MSE emits a signal via one of its output ports, the signal goes along the preset path and is inverted in every MSE throughout the path. If an MSE has 1 as its input signal, it is mapped to the upper crossbar switch, while if the input signal is -1, it is

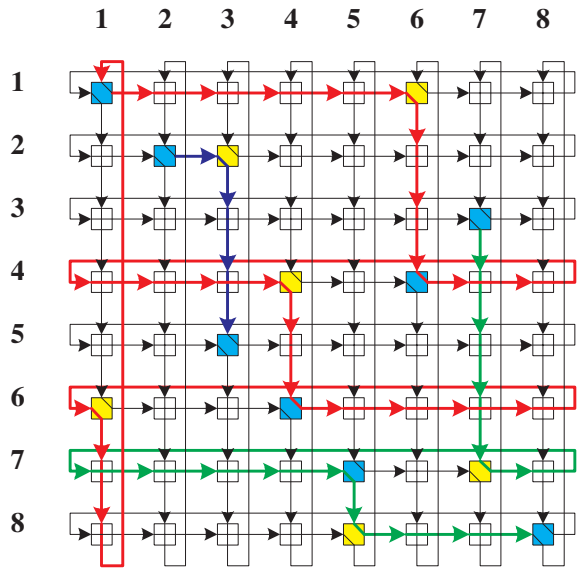


Fig. 8. A hardware implementation of the proposed mapping algorithm: A crossbar mapper

mapped to the lower crossbar switch. When the signal visits an MSE twice, it stops there because a partial assignment has been found. However, to find a complete solution, the MSEs from which a signal starts have to be selected externally for each partial assignment.

In a more general case of $k > 2$ and $m > 2$, a (k, m)-dim crossbar can be replaced by $max\{k, m\}$ crossbar switches.

Theorem 6: *Each input-output pair matched by the SSA algorithm for a (k, m)-dim crossbar can be mapped to $max\{k, m\}$ crossbar switches with no conflict if $k > 2$ and $m > 2$.*

Proof: The SSA algorithm for an MIOQ switch with a (k, m)-dim crossbar selects at most k of N input buffers to send packets and at most m of M output buffers to receive packets in a time slot. Therefore, all input-output pairs matched by the SSA algorithm can be mapped to a Clos network which is composed of three stage switches: N first-stage switches of size $k \times n$, n second-stage switches of size $N \times N$, and N third stage switches of size $n \times m$. According to the Slepian-Duguid Theorem, this mapping is rearrangeable if and only if $n \geq max\{k, m\}$. An MIOQ switch with n crossbar switches in parallel includes this Clos network topologically. (The MIOQ switch is composed of N first-stage switches of size $N \times n$, n second-stage switches of size $N \times N$, and N third stage switches of size $n \times M$.) Therefore, all input-output pairs matched by the SSA algorithm can be mapped rearrangeably to an MIOQ switch which has more than $max\{k, m\}$ crossbar switches in parallel. $\diamond$

Although the proposed mapping algorithm is not applicable for $k > 2$ and $m > 2$, the rearrangement process in the Paull's Theorem can be used to find a mapping with no conflict.

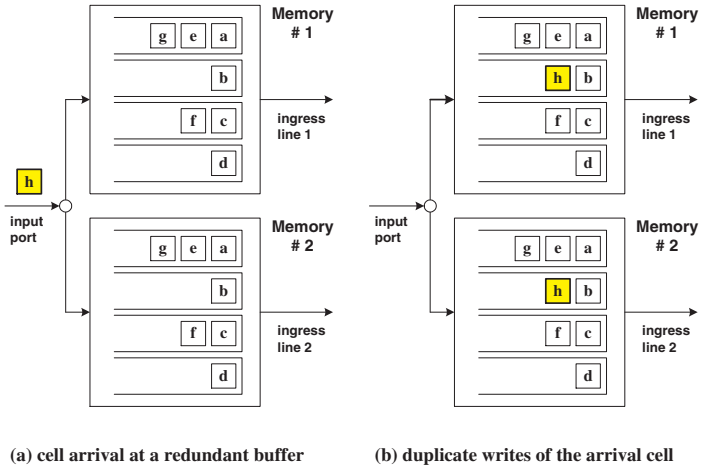


Fig. 9. An example of write operation of a redundant buffer

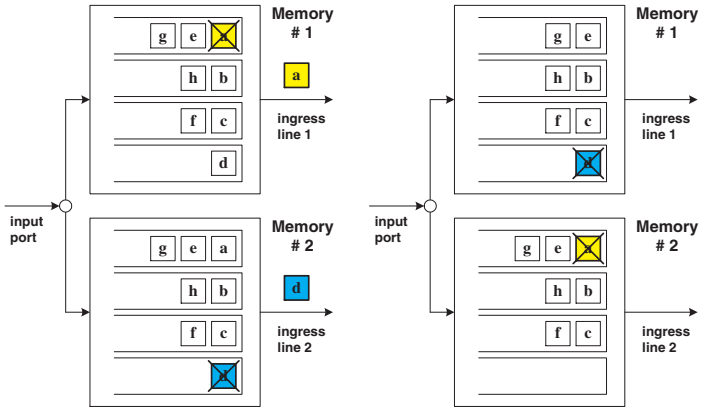


Fig. 10. An example of read operation of a redundant buffer

VI. Implementation of Mutiple Input Buffers with a Redundant Buffer

For the emulation of output queueing, the MIOQ switch requires any two of N input buffers to be accessed simultaneously because the switch has two ingress lines. This means that N input buffers cannot be implemented logically on a single memory as in [27]. A straightforward approach for implementing N independent buffers is to use N physical memories. However, as a natural result, this approach has some drawbacks: First, the area and power requirements of a line card which contains N input buffers increase. Second, memory-space sharing between buffers is impossible, which leads to inefficient usage of finite memory space.

To avoid these problems of the straightforward approach, we propose a buffering scheme called *redundant buffering*. A redundant buffer for an MIOQ switch to emulate an output-

TABLE I

COMPARISON OF OUTPUT QUEUEING EMULATION SCHEMES

	Speedup Factor	**# of Parallel Switches**
OQ Switch	N	1
CIOQ Switch	2	1
PPS with OQ Switches	N	3
PPS with CIOQ Switches	2	3
PPS with CIOQ Switches	1	6
MIOQ Switch	**1**	**2**

queued switch is composed of two physical memories. Each of two memories in a redundant buffer contains N logically-separated input buffers as in [27]. In a redundant buffer, two memories maintain the same contents redundantly, in other words, each memory of a redundant buffer has all cells in N input buffers. As a result, redundant buffering scheme requires double the amount of conventionally-used memory, which is a drawback of this scheme. However, compared with the cell arrival rate at each input, a $(2, 2)$-dimensional crossbar fabric provides sufficient bandwidth transferring cells from inputs to outputs, which means that most of the cells in an MIOQ switch are stored in output buffers instead of input buffers. Therefore, in an MIOQ switch, large memory space is not required for input buffers, and thus, the memory-space penalty of the redundant buffering scheme is not critical in practice. Hence, in an MIOQ switch with redundant buffers, when two cells have to be read simultaneously from two different input buffers at an input, each cell can be read from one of two memories, respectively.

Figure 9 and 10 illustrate write and read operations of a redundant buffer which is composed of two memories: Memory 1 and Memory 2. As in Figure 9, when a new cell h arrives at a redundant buffer, both memories write the cell in the same location. When two cells, cell a and cell d, are required to be read from two different buffers as in Figure 10, cell a and d are read from Memory 1 and 2, respectively. After the read operation, the cell which has been read from one memory has to be eliminated in the other memory. This operation can be done logically with no memory access by updating the *pointer register* which points to the corresponding cells as in [27]. Note that a redundant buffer does not need interconncetion networks to connect with ingress lines and as in Figure 10, memories of a redundant buffer are directly connected to ingress lines one by one.

VII. CONCLUSIONS

In recent years, there have been many research works focused on the emulation of output queueing with a limited-speedup switch. In Table I, we summarize and compare the various schemes for output queueing emulation including the CIOQ switch and the PSAs. As remarked in Introduction, a CIOQ switch with the SMM algorithm requires a speedup of two on a crossbar switch for output queueing emulation. Also, PPS with six parallel switches each of which is a CIOQ switch with the SMM algorithm can emulate an output-queued switch exactly without any speedup.

As an alternative, in this paper, we show that the MIOQ switch with a $(2, 2)$-dim crossbar, which can be implemented with a $2N \times 2N$ crossbar switch, can emulate an output-queued switch exactly with no speed. This result holds for all arrival traffic patterns, any size of switches and a broad class of service scheduling algorithms including FIFO, WFQ and strict priority queueing. We also show that a modified version of the MIOQ switch composed of two $N \times N$ crossbar switches in parallel instead of a $(2, 2)$-dim crossbar can achieve the same result. In addition, we propose a buffering scheme that requires two physical memories to implement N input buffers, where two of them can be accessed simultaneously. Consequently, we conclude that two crossbar switches in parallel and two physical memories at each input are sufficient for an MIOQ switch to emulate an output-queued switch.

Although the result reported in this paper is an advanced one compared to previous works, it also requires a quite complex scheduling algorithm, i.e., the SSA algorithm, which has the similar complexity to that of the SSM algorithm used for an CIOQ switch. Considering the running-time requirement of fast switches and routers, it is still not practical. However, this result does not preclude the existence of other algorithms that can be implemented at higher speeds. We believe that this will continuously be an important topic for future research.

REFERENCES

[1] GSR 12000 Router, Cisco Systems.
Available: http://www.cisco.com/warp/public/cc/pd/rt/12000/

[2] PI40 Fabric Chip Set, Lucent Technologies. Available: http://www.agere.com/netcom/docs/PB01136.pdf

[3] N. Mckeown, "Scheduling Algorithms for Input-queued Cell Switches," *Ph. D. dissertation*, Univ. California at Berkeley, 1995.

[4] N. Mckeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking*, Vol. 7, No. 2, pp. 188-201, April 1999.

[5] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% Throghput in an Input-Queued Switch," *IEEE Transactions on Communications*, Vol. 47, No. 8, pp. 1260-1267, Aug. 1999.

[6] R. O. LaMaire and D. N. Serpanos, "Two Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues," *IEEE/ACM Trans. Networking*, Vol. 2, pp. 471-482, Oct. 1994.

[7] H. J. Chao, "Saturn: A Terabit Packet Switch using Dual Round-Robin," *IEEE Comunnications Magazine*, Vol. 38, pp. 78-84, Dec. 2000.

[8] D. Ferrari and D. C. Verma, "A Scheme for Real-time Channel Establishment in Wide-Area Network," *IEEE J. on Selected Areas Commun.*, Vol. 8, No. 3, pp. 368-379, April, 1990.

[9] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach tp Flow Control in Integrated Services Networks: The Sing Node Case," *IEEE/ACM Trans. on Networking*, Vol 1, pp. 344-357, June, 1993.

[10] H. Zhang, "Service Disciplines for Guaranteed Perfromance Service in Packet-Switched Networks," *Proceedings of the IEEE*, 83(10), pp. 1374-1399, October, 1995.

[11] J. C. R. Bennet and H. Zhang, "WF^2Q: Worst-case Fair Weighted Fair Qeueuing," *IEEE INFOCOM'96*, pp. 120-128.

[12] N. McKeown, B. Prabhakar and M. Zhu, "Matching Output Queueing with Combined Input and Output Queueing," *Proc. 35th Annual Allerton Conf. on Communications, Control and Computing*, October, 1997.

[13] B. Prabhakar and N. McKeown, "On the Speedup Required for Combined Input and Output Queued Switching," *Computer Systems Lab. Technical Report*, CSL-TR-97-738, Stanford University.

[14] R. Guerin and K. N. Sivarajan, "Delay and Throughput Perfromance of Speed-Up Input-Queueing Packets Switches," *IBM Research Report*, RC 20892, June, 1997.

[15] P. Krishna, N. S. Patel, A. Charny and R. Simcoe, "On the Speedup Required for Work-Conserving Crossbar Switches," in *6th Int. Workshop Quality of Service (IWQoS'98)*, pp. 225-234, Napa, CA, May, 1998.

[16] E. Leonardi, M. Mellia, F. Neri and M. A. Marson, "On the Stability of Input-Queued Switches with Speed-Up," *IEEE/ACM Transactions on Networking*, Vol. 9, No. 1, Feb. 2001.

[17] S. T. Chuang, A. Goel, N. McKeown and B. Prabhakar, "Matching Output Queueing with Combined Input and Output Qeueing," *IEEE J. Select. Areas Commun.*, Vol. 17, pp. 1030-1039, Dec. 1999.

[18] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," *Amer. Math. Monthly*, Vol. 69, pp. 9-15, 1962.

[19] S. Iyer, A. A. Awadallah and N. McKeown, "Analysis of a Packet Switch with Memories Running Slower than the Line Rate," *IEEE INFOCOM'2000*, Vol. 2, pp. 529-537 , March, 2000.

[20] S. Iyer and N. McKeown, "Making Parallel Packet Switches Practical," *IEEE INFOCOM'2001*, Vol. 3, pp. 1680-1687, March, 2001.

[21] S. S. Mneimneh, V. Sharma and K. Y. Siu, "On Scheduling Using Parallel Input-Output Queued Crossbar Switches with no Speedup," *IEEE Workshop on High Performance Switching and Routing*, pp. 317-323, 2001.

[22] H. I. Lee and S. W. Seo, "A Practical Approach for Statistical Matching of Output Queueing," *IEEE Journal of Selected Areas in Communications*, Vol. 21, No. 4, pp. 616-629, May 2003.

[23] Y. S. Yeh, M. G. Hluckyj, and A. Acampora, "The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching," IEEE J. on Select. Areas in Commun., Vol. 5, pp. 1274-1283, Oct. 1987.

[24] R. Bumcrot, P. Campbell, J. Gallian, C. Arney and W. Meyer, *Principles and Practice of Matchematics*, Springer Verlag New York.

[25] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press Cambridge.

[26] J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers.

[27] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Trans. on Computers*, Vol. 41, No. 6, June 1992.

[28] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, New York: Academic, 1965.