Birkhoff-von Neumann Input Buffered Crossbar Switches

Cheng-Shang Chang   Wen-Jyh Chen   Hsiang-Yi Huang
Dept. of Electrical Engineering
National Tsing Hua University
Hsinchu 30043 Taiwan, R.O.C.
Email: cschang@ee.nthu.edu.tw
{wjchen, huhuang}@hardy.ee.nthu.edu.tw

Abstract— **In our previous paper [5], we proposed a scheduling algorithm that is capable of providing rate guarantees for input-buffered crossbar switches. The algorithm is based on a decomposition result by Birkhoff and von Neumann for a doubly substochastic matrix. An input buffered crossbar switch that uses such an algorithm is called the Birkhoff-von Neumann switch in this paper. For the Birkhoff-von Neumann switch, the rate guarantees are uniformly good for all non-uniform traffic, and it does not require framing or internal speedup. Our objective of this paper is to make the Birkhoff-von Neumann switch more complete and practical. We do so by addressing three topics: providing best-effort services in the Birkhoff-von Neumann switch, hardware implementation of the switch fabric, and multistage Birkhoff-von Neumann switches.**

Keywords— **input buffered switches, crossbar switches, rate guarantees, scheduling, generalized processor sharing, Banyan networks, rearrangeable networks**

I. Introduction

Due to their scalability, input buffered crossbar switches have received a lot of attention recently (see e.g., [27], [21], [17], [7], [22], [6], [15], [28], [11], [18]). An input buffered crossbar switch with N input ports and N output ports has a segregated buffer for each input port. In such a switch, time is slotted and synchronized so that packets from different input buffers can be read out simultaneously within a time slot. In a time slot, a crossbar switch sets up a connection pattern corresponding to a permutation matrix (see Figure 1). As a permutation matrix is a one-to-one mapping from input ports to output ports, packets destined to the same output ports cannot be transmitted at the same time. As discussed in [22], such limitation causes two potential problems: low throughput due to head-of-line (HOL) blocking and the difficulty of controlling packet delay. By allowing the switch to transmit non-HOL packets, it is shown in [21], [22] that 100% throughput can be achieved by certain weighted matching algorithms. To control packet delays, it is shown in [7], [28], [6], [15] that certain deterministic guarantees, such as queue length and delay, can be obtained by internal speedup. In particular, a speedup of 2 is sufficient to mimic the behavior of an output-buffered switch (see [7], [28]).

This research is supported in part by the National Science Council, Taiwan, R.O.C., under Contracts NSC-88-2213-E007-004 and NSC-87-2219-E007-005.

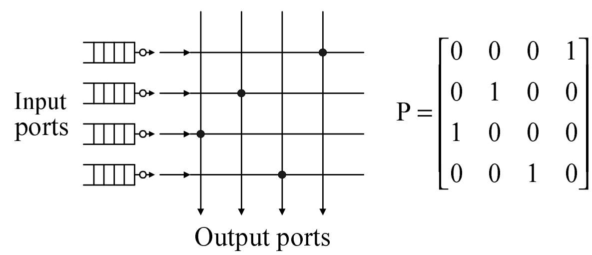

Fig. 1. A 4×4 input-buffered crossbar switch

In the recent paper [11], Hung, Kesidis and McKeown used an idling weighted round robin (WRR) algorithm in [1] to achieve rate guarantee for each input-output pair without internal speedup. This in turn implies 100% throughput (if the rate for each input-output pair is known). Similar approaches are also addressed by Lee and Lam [17] and Li and Ansari [18]. As the usual WRR algorithm, such an approach requires to choose a frame size. A large frame size implies a large worst case packet delay and a large memory requirement for the storage of all the connection patterns in a frame. On the other hand, a small frame size implies a large rate granularity (the minimum rate allocated to an input-output pair). As a result, the WRR algorithm fails to provide uniform rate guarantees for all non-uniform traffic due to its framing requirement.

Based on a decomposition result by Birkhoff and von Neumann for a doubly substochastic matrix, we proposed in our previous paper [5] a scheduling algorithm that is capable of providing rate guarantees for input-buffered crossbar switches. For this approach, the rate guarantees are uniformly good for all non-uniform traffic, and it does not require framing or internal speedup. An input buffered crossbar switch that uses the Birkhoff-von Neumann decomposition algorithm is called the Birkhoff-von Neumann switch in this paper. Our objective of this paper is to make the Birkhoff-von Neumann switch more complete and practical. We do so by addressing three topics: providing best-effort services in the Birkhoff-von Neumann switch, hardware implementation of the switch fabric, and multistage Birkhoff-von Neumann switches.

The paper is organized as follows: in Section II, we review the Birkhoff-von Neumann decomposition algorithm and illustrate how rate guarantees can be achieved. In Section III, we formulate the problem of providing best-effort services as an optimization problem for efficiency and fairness. We present an on-line algorithm for measuring rate demands and report simulation results of the optimization approach. In Section IV, we address the problem encountered in hardware implementation and report our implementation on a FPGA chip. In Section V, we illustrate how one builds a larger switch by interconnecting smaller switches. Two types of connecting networks are addressed: a two-stage Banyan network and a three-stage rearrangeable network. The paper is concluded in Section VI.

II. Guaranteed-rate services

In this section, we show how one provides rate guarantees for all non-uniform traffic in input buffered crossbar switches. The main idea, as presented in our previous paper [5], is to use the capacity decomposition approach by Birkhoff [4] and von Neumann [29]. To explain the idea, let $R = (r_{i,j})$ be the rate matrix with $r_{i,j}$ being the rate requested by the traffic from input i to output j for an $N \times N$ input-buffered crossbar switch. Then under the following two conditions

$$\sum_{i=1}^{N} r_{i,j} \leq 1, \ \ \forall j, \quad \text{and} \tag{1}$$

$$\sum_{j=1}^{N} r_{i,j} \leq 1, \ \ \forall i, \tag{2}$$

there exists a set of positive numbers ϕ_k and permutation matrices P_k, $k = 1, \ldots, K$ for some $K \leq N^2 - 2N + 2$ that satisfies

$$R \leq \sum_{k=1}^{K} \phi_k P_k, \quad \text{and} \tag{3}$$

$$\sum_{k=1}^{K} \phi_k = 1. \tag{4}$$

The two conditions in (1) and (2), called "no overbooking" conditions in [11], are intuitively clear as they simply state that neither the total rate to an output port nor the total rate coming out from an input port can be larger than 1. Such a decomposition may be obtained off-line and it only needs to be recomputed when the requested rates change. Once we obtain such a decomposition, we can simply schedule the connection pattern P_k proportional to its weight ϕ_k, $k = 1, \ldots, K$. One on-line scheduling algorithm to do this is the Packetized Generalized Processor Sharing (PGPS) algorithm in Parekh and Gallager [25] (or the Weighted Fair Queueing (WFQ) in Demers, Keshav, and Shenkar [8]).

A. The decomposition algorithm

In this section, we describe the Birkhoff-von Neumann algorithm to obtain the decomposition in (3).

A nonnegative matrix $R = (r_{i,j})$ that satisfies the conditions in (1) and (2) is known to be *doubly substochastic* (see e.g., Marshall and Olkin [19]). If, furthermore, both inequalities in (1) and (2) are equalities, then the matrix $R = (r_{i,j})$ is called *doubly stochastic.* The decomposition basically consists of two steps: (i) it first finds a doubly stochastic matrix that is not smaller than the original rate matrix, and (ii) it then finds a decomposition for the doubly stochastic matrix obtained from the first step. The first step is based on the following result by von Neumann.

Proposition 1 (von Neumann [29]) *If a matrix $R = (r_{i,j})$ is doubly substochastic, then there exists a doubly stochastic matrix $\tilde{R} = (\tilde{r}_{i,j})$ such that*

$$r_{i,j} \leq \tilde{r}_{i,j}, \ \ \forall i, j.$$

This can be constructed by the following algorithm.
Algorithm 1:
(i) If the sum of all the elements in R is less than N, then there exists an element (i, j) such that $\sum_n r_{i,n} < 1$ and $\sum_m r_{m,j} < 1$.
(ii) Let $\epsilon = 1 - \max[\sum_n r_{i,n}, \sum_m r_{m,j}]$. Add ϵ to the element in the $(i, j)^{th}$ position to obtain a new matrix R_1. Then in R_1, the number of row sums and column sums that are strictly smaller than 1 is at least one less than that in R.
(iii) Repeat this procedure until the sum of all the elements is equal to N. In this case, we derive a doubly stochastic matrix.
Clearly, the above algorithm will be ended after at most $2N - 1$ steps as the last step will make one row sum and one column sum to 1. As the matrix is $N \times N$, the computational complexity of finding an element (i, j) with $\sum_n r_{i,n} < 1$ and $\sum_m r_{m,j} < 1$ is $O(N^2)$. Thus, the computational complexity of Algorithm 1 is $O(N^3)$.

The second step is based on Birkhoff's result on doubly stochastic matrices [4].

Proposition 2 (Birkhoff [4]) *For a doubly stochastic matrix $\tilde{R}$, there exists a set of positive numbers ϕ_k and permutation matrices P_k such that*

$$\tilde{R} = \sum_k \phi_k P_k.$$

Let e be the column vector with all its elements being 1. As $\tilde{R}$ is doubly stochastic,

$$e = \tilde{R}e = \sum_k \phi_k(P_k e) = (\sum_k \phi_k)e.$$

This shows that

$$\sum_k \phi_k = 1. \tag{5}$$

Basically, Proposition 2 says that doubly stochastic matrices are within the convex hull of the permutation matrices.

In the following, we present an algorithm to obtain the set of positive numbers ϕ_k and permutation matrices P_k. As indicated in [19], the algorithm was originated by Dulmage and Halperin [9]. We note that the same algorithm was rediscovered by Inukai [12] for the time slot assignment problem in a satellite-switched time-division multiple access (SS/TDMA) system.

Algorithm 2:

(i) For a doubly stochastic matrix $\tilde{R}$, let $(i_1, i_2, \ldots, i_N)$ be a permutation of $(1, 2, \ldots, N)$ such that $\prod_{k=1}^N \tilde{r}_{k,i_k} > 0$. The existence of such a permutation for a doubly stochastic matrix is given by Mirsky [23], pp. 185 and Berge [2], pp. 105.

(ii) Let P_1 be the permutation matrix corresponding to $(i_1, i_2, \ldots, i_N)$, and $\phi_1 = \min_{1 \le k \le N}[\tilde{r}_{k,i_k}]$. Define the matrix R_1 by

$$R_1 = \tilde{R} - \phi_1 P_1.$$

(iii) If $\phi_1 = 1$, then

$$R_1 e = \tilde{R}e - P_1 e = \mathbf{0},$$

where $\mathbf{0}$ is the column vector with all its elements being zero. In this case, the matrix R_1 is the zero matrix as all the column sums are zero. Thus, we have completed the representation.

(iv) If $\phi_1 < 1$, then the matrix $\frac{1}{1-\phi_1}R_1$ is doubly stochastic and we can continue the decomposition from (i).

As each iteration reduces at least one element to zero, it is easy to see that the algorithm will be stopped by at most $N^2 - N + 1$ steps (the last one will reduce N elements to zero). In fact, as shown in [19] and references therein, the algorithm will be stopped by at most $N^2 - 2N + 2$ steps. The number $N^2 - 2N + 2$ cannot be reduced by a smaller number for the worse case. The main computation complexity of Algorithm 2 is to find a permutation $(i_1, i_2, \ldots, i_N)$ such that $\prod_{k=1}^N \tilde{r}_{k,i_k} > 0$. This can be done by converting the matrix $\tilde{R}$ into a $0-1$ valued adjacency matrix of a bipartite graph and finding a maximum matching in that bipartite graph (in this case, a maximum matching is ensured to be perfect). It is known (see e.g., [24], Theorem 10.2) that the computational complexity for the bipartite matching problem is $O(N^3)$ via the alternating path algorithm, and can be improved to $O(N^{2.5})$ via mapping the problem to the maximum flow problem. Thus, the computational complexity of Algorithm 2 is $O(N^{4.5})$.

We note that if there is an integer f such that the matrix $f\tilde{R}$ contains all integer-valued elements, then Algorithm 2 will be stopped by at most f steps. To see this, observe that

$$\frac{1}{1-\phi_1}R_1 = \frac{\tilde{R} - \phi_1 P_1}{1-\phi_1} = \frac{f\tilde{R} - f\phi_1 P_1}{f - f\phi_1}.$$

For $\phi_1 < 1$, the matrix $(f - f\phi_1)(\frac{1}{1-\phi_1}R_1)$ is a matrix with all integer-valued elements. Since $f\phi_1$ is an integer not less than 1, we only need to multiply the matrix after one step by a smaller number than f for it to become a matrix with all integer-valued elements. Thus, Algorithm 2 will be stopped by at most f steps. This shows that the WRR algorithm in [11], [17] is a special case of Algorithm 2 when the elements in the matrix $\tilde{R}$ are rational numbers. Besides, Algorithm 2 generates a more compact representation than the Slepian-Duguid algorithm used in the WRR algorithm.

B. The on-line scheduling algorithm

In this section, we present our on-line scheduling algorithm. From the constructions in Algorithms 1 and 2, we can find a set of positive numbers ϕ_k and permutation matrices P_k, $k = 1, \ldots, K$ for some $K \le N^2 - 2N + 2$ that satisfies

$$R \le \sum_{k=1}^K \phi_k P_k, \quad \text{and} \tag{6}$$

$$\sum_{k=1}^K \phi_k = 1. \tag{7}$$

In view of (6) and (7), the amount of time that the crossbar switch sets its connection pattern according to the permutation matrix P_k should be proportional to ϕ_k. As time is slotted, we can use the Packetized Generalized Processor Sharing (PGPS) algorithm in Parekh and Gallager [25] (or the Weighted Fair Queueing (WFQ) in Demers, Keshav, and Shenkar [8]) to approximate the scheme.

Algorithm 3:

(i) Assign a class of tokens for each permutation matrix P_k, $k = 1, \ldots, K$.

(ii) At time 0, generate a token for each class. Assign the virtual finishing time of the first class k token by

$$F_k^1 = \frac{1}{\phi_k}.$$

Sort these K tokens in an increasing order of the virtual finishing times.

(iii) The switch serves the tokens in an increasing order of the virtual finishing times. Once a class k token is served in a time slot, the switch sets its connection pattern according to the permutation matrix P_k for that time slot.
(iv) After the $(\ell-1)^{th}$ class k token is served, the switch generates the ℓ^{th} class k token and assign the virtual finishing time by

$$F_k^\ell = F_k^{\ell-1} + \frac{1}{\phi_k}.$$

Insert the newly generated token in the sorted token list and repeat the algorithm from (iii).

Note that Algorithm 3 is in fact a special case of PGPS. It corresponds to the case that there is an infinite number of packets at time 0 for PGPS. As the algorithm generates a new token of the same class after a token is served, there are exactly K classes of tokens in the sorted token list. Since the tokens are always backlogged, there is no need to keep track of the virtual time in [25]. Thus, Algorithm 3 is independent of the buffer contents at any time.

The complexity of Algorithm 3 is to insert every new token into the sorted token list. As there are K tokens in the sorted list and $K \le N^2 - 2N + 2$, the computational complexity of Algorithm 3 is $O(\log N)$. Since it takes $N \log_2 N$ bits to describe a permutation matrix and there are $O(N^2)$ permutation matrices needed to be stored in Algorithm 3, the memory complexity of Algorithm 3 is $O(N^3 \log N)$.

Let $C_{i,j}(t)$ be the cumulative number of slots that are assigned to the traffic from input i to output j by time t. Also, let $p_{k,i,j}$ be the element at the (i,j) position of the permutation matrix P_k obtained from Algorithm 2. Note that $p_{k,i,j}$ only takes values 0 and 1. Define $E_{i,j} = \{k : p_{k,i,j} = 1\}$ to be the subset of $\{1, 2, \ldots, K\}$ such that for all k in $E_{i,j}$, the permutation matrix P_k has a nonzero element at the (i,j) position. The following theorem states the rate guarantee result. Its proof can be found in our previous paper [5].

Theorem 3 *For any rate matrix $R = (r_{i,j})$ that satisfies (1) and (2), Algorithms 1,2 and 3 generate a scheduling policy that guarantees*

$$\begin{aligned}\sum_{k\in E_{i,j}} \phi_k(t-s) - s_{i,j} &\le C_{i,j}(t) - C_{i,j}(s) \\ &\le \sum_{k\in E_{i,j}} \phi_k(t-s) + s_{i,j}, \end{aligned} \tag{8}$$

where

$$s_{i,j} = \min[K, |E_{i,j}| + \sum_{k\in E_{i,j}} \phi_k(K-1)], \tag{9}$$

and $|E_{i,j}|$ is the number of elements in the set $E_{i,j}$.

Note that K and ϕ_k's are the parameters obtained from Algorithm 2. As $K \le N^2 - 2N + 2$ in Algorithm 2, we have $s_{i,j} \le N^2 - 2N + 2$. Also, we have from (6) that

$$\sum_{k\in E_{i,j}} \phi_k = \sum_{k=1}^{K} \phi_k p_{k,i,j} \ge r_{i,j}. \tag{10}$$

This in turn implies

$$C_{i,j}(t) - C_{i,j}(s) \ge r_{i,j}(t-s) - N^2 + 2N - 2,$$

for all i, j, and $s \le t$.

For example, consider the following rate matrix

$$R = \begin{bmatrix} 0 & 0.3 & 0.2 & 0.4 \\ 0.2 & 0.3 & 0 & 0.2 \\ 0.4 & 0.1 & 0.3 & 0 \\ 0.2 & 0 & 0.2 & 0.3 \end{bmatrix}. \tag{11}$$

Using Algorithm 1 with the sequence of pivots $(1,2), (2,1), (2,2), (3,2), (3,3), (4,3), (4,4)$, we have the following doubly stochastic matrix

$$\tilde{R} = \begin{bmatrix} 0 & 0.4 & 0.2 & 0.4 \\ 0.4 & 0.4 & 0 & 0.2 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0.2 & 0 & 0.4 & 0.4 \end{bmatrix}. \tag{12}$$

From Algorithm 2, we have the following decomposition

$$\begin{aligned}\tilde{R} = 0.4 \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + 0.4 \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ + 0.2 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{13}$$

Using (9), we can find $S = (s_{i,j})$ as follows:

$$S = \begin{bmatrix} 0 & 1.8 & 1.4 & 1.8 \\ 1.8 & 1.8 & 0 & 1.4 \\ 1.8 & 1.4 & 1.8 & 0 \\ 1.4 & 0 & 1.8 & 1.8 \end{bmatrix}. \tag{14}$$

III. Best-effort services

In addition to the traffic with guaranteed rates, there is best-effort traffic. Clearly, if the total traffic fed into the Birkhoff-von Neumann switch satisfies the "no overbooking" conditions, then there is nothing that needs to be done. However, in practice only the guaranteed-rate traffic can be assumed to satisfy the "no overbooking" conditions via appropriate admission control. As there is no admission control for best-effort traffic, the total traffic might violate the "no overbooking" conditions. When this happens, it becomes a resource sharing problem.

A. Problem formulation

To formulate the resource sharing problem, let $R^g = (r^g_{i,j})$ be the rate matrix for the guaranteed-rate services and $R = (r_{i,j})$ be the allocated rate matrix. Via admission control, we assume that R^g satisfies the "no overbooking" conditions. We also assign the guaranteed-rate services higher priority than the best-effort services. This implies that the allocated rates should be at least the rates of the guaranteed-rate services, i.e., $r_{i,j} \geq r^g_{i,j}$ for all i, j. One may then choose a utility function $U(r_{i,j}, i, j = 1, \ldots, N)$ for optimizing the resource sharing problem. As the matrix R is clearly constrained by the "no overbooking" conditions in (1) and (2) in the Birkhoff-von Neumann switch, we have the following optimization problem

$$\begin{aligned}
\text{Maximize} \quad & U(r_{i,j}, i, j = 1, \ldots, N) \\
\text{Subject to} \quad & r^g_{i,j} \leq r_{i,j}, \ i, j = 1, 2, \ldots, N, && (15)\\
& \sum_{i=1}^{N} r_{i,j} \leq 1, \ j = 1, 2, \ldots, N, && (16)\\
& \sum_{j=1}^{N} r_{i,j} \leq 1, \ i = 1, 2, \ldots, N. && (17)
\end{aligned}$$

We note that unlike the distributed setting in [20], the optimization problem is centralized as all the traffic information is available to the switch.

The utility function is usually chosen to accomplish two objectives: *efficiency* and *fairness*. We address the issue on efficiency first. Let $R^e = (r^e_{i,j})$ be the rate matrix for the total traffic (we will discuss how to obtain an estimate for such a rate matrix in the next section). As the total traffic includes the traffic from the guaranteed-rate services, we have $r^e_{i,j} \geq r^g_{i,j}$. Intuitively, one may view $r^e_{i,j}$ as the potential rate demand for the traffic from the i^{th} input port to the j^{th} output port. As $r_{i,j}$ is the allocated rate for the traffic from the i^{th} input port to the j^{th} output port, the throughput from the i^{th} input port to the j^{th} output port is $\min[r^e_{i,j}, r_{i,j}]$. Thus, the total throughput is $\sum_{i=1}^{N}\sum_{j=1}^{N} \min[r^e_{i,j}, r_{i,j}]$ and it can be used as a utility function. To maximize the total throughput, we then have to maximize $\sum_{i=1}^{N}\sum_{j=1}^{N} \min[r^e_{i,j}, r_{i,j}]$ under the constraints in (15)-(17).

As $r^e_{i,j} \geq r^g_{i,j}$, allocating $r_{i,j}$ more than the rate demand $r^e_{i,j}$ is as good as allocating $r^e_{i,j}$. Thus, the above nonlinear optimization problem is equivalent to the following linear programming problem:

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} r_{i,j} \\
\text{Subject to} \quad & r^g_{i,j} \leq r_{i,j} \leq r^e_{i,j}, \quad i, j = 1, 2, \ldots, N, \\
& \sum_{i=1}^{N} r_{i,j} \leq 1, \ j = 1, 2, \ldots, N, \\
& \sum_{j=1}^{N} r_{i,j} \leq 1, \ i = 1, 2, \ldots, N.
\end{aligned}$$

Such a linear programming problem can be easily mapped to a maximum flow problem. In Figure 2, we show the corresponding maximum flow problem when there is no guaranteed-rate traffic, i.e., $r^g_{i,j} = 0$ for all i, j.

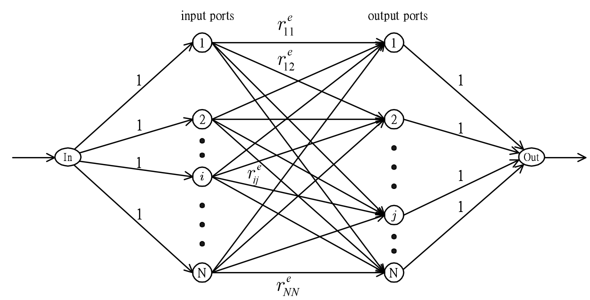

Fig. 2. The maximum flow problem

The maximum flow problem can be further simplified by making the following observation: the potential rate demand from an input port with rate 1 cannot be larger than 1, i.e., $\sum_{j=1}^{N} r^e_{i,j} \leq 1$ for all $i = 1, \ldots, N$. This implies the constraints $\sum_{j=1}^{N} r_{i,j} \leq 1$, $i = 1, 2, \ldots, N$, are redundant, and they can be removed. In view of this, we can decompose the maximum flow problem into N subproblems as follows: for $j = 1, \ldots, N$,

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{N} r_{i,j} \\
\text{Subject to} \quad & r^g_{i,j} \leq r_{i,j} \leq r^e_{i,j}, \quad i = 1, 2, \ldots, N, \\
& \sum_{i=1}^{N} r_{i,j} \leq 1.
\end{aligned}$$

The maximum flow for the j^{th} subproblem is simply $\min[\sum_{i=1}^{N} r^e_{i,j}, 1]$ as the total rate demand for the j^{th} output port is $\sum_{i=1}^{N} r^e_{i,j}$ and the rate supply for the j^{th} output port is 1. However, the solution of such a maximum flow problem may not be unique. This leads us to consider the second objective "fairness."

Here we consider the "max-min fairness." The rate allocation under the max-min fairness can be computed by the following "water filling" procedure (see e.g. [20], [3]): for the j^{th} subproblem, the (i, j) pair is initially allocated the rate $r^g_{i,j}$, $i = 1, \ldots, N$. Then rate allocations of all pairs grow linearly. When a pair reaches its associated rate demand limitation, it stops. The other pairs proceed the linear filling until the rate supply runs out. Clearly, such a "water filling" procedure achieves the maximum flow for each subproblem. It also maximizes the minimum rate allocation under the following constraints

$$r^g_{i,j} \le r_{i,j} \le r^e_{i,j}, \qquad i = 1, 2, \ldots, N,$$
$$\sum_{i=1}^{N} r_{i,j} = \min[\sum_{i=1}^{N} r^e_{i,j}, 1].$$

In fact, it can be shown that the rate allocation obtained from the "water filling" procedure also achieves the most "balanced" allocation in the sense of majorization ordering [19] and thus minimizes all the utility functions that are Schur convex. As the family of symmetric convex functions are Schur convex, proportional fairness addressed in [13], [14] is also achieved by the "water filling" procedure.

B. On-line measurement

We still have to obtain an estimate for the rate demand $r^e_{i,j}$. To do this, we first choose a measuring period T that is long enough to perform the Birkhoff-von Neumann decomposition algorithm. Let $A_{i,j}(t)$ be the cumulative number of arrivals by time t for the traffic from the i^{th} input to the j^{th} output port. In the n^{th} measuring period, we measure the number of arrivals for the traffic from the i^{th} input port to the j^{th} output port, i.e., $A_{i,j}(nT) - A_{i,j}((n-1)T)$. Form the following recursive estimate

$$r^e_{i,j}(n+2) = (1-\alpha(n)) r^e_{i,j}(n+1) + \alpha(n)\Big(\frac{A_{i,j}(nT) - A_{i,j}((n-1)T)}{T}\Big), \qquad (18)$$

$$r^e_{i,j}(0) = r^e_{i,j}(1) = \frac{1}{N}, \qquad (19)$$

for some $0 < \alpha(n) < 1$ and $n \ge 1$. Here the initial estimates in (19) are chosen to be uniform and there is one period lag due to computation needed for the Birkhoff-von Neumann decomposition algorithm. A timing diagram is shown in Figure 3. The constant $\alpha(n)$ depends on the traffic from the best-effort services. If the traffic is very dynamic, the constant $\alpha(n)$ should be large for easy adaptation. On the other hand, if the traffic is smooth, the constant $\alpha(n)$ should be small. In fact, if we choose $\alpha(n)$ to be $\frac{1}{n}$, then we have

$$r^e_{i,j}(n+2) = \frac{1}{n} A_{i,j}(nT)$$

which is the sample mean of the measured traffic.

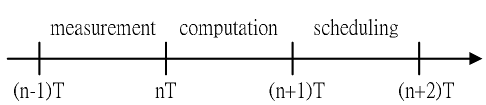

Fig. 3. A timing diagram for the recursive estimate

Note that $\sum_{j=1}^{N} A_{i,j}(nT) - A_{i,j}((n-1)T)$ is the number of arrivals at the i^{th} input port within a measuring period. As there is at most one arrival in a time slot,

$$\frac{A_{i,j}(nT) - A_{i,j}((n-1)T)}{T} \le 1.$$

From induction, it follows from (18) that

$$\sum_{j=1}^{N} r^e_{i,j}(n) \le 1$$

for all n. This establishes the claim needed for the decomposition of the maximum flow problem in the previous section.

To understand the performance of the best-effort services in the Birkhoff-von Neumann switch, we perform various simulations for 4×4 switches. In these simulations, we assume there is no guaranteed-rate services, i.e., $r^g_{i,j} = 0$ for all i, j. The best-effort traffic is generated as follows: at each time slot, a packet is generated independently with probability ρ at each input port. The destination of the generated packet is uniformly distributed among all the four output ports. For each link utilization ρ, we average the queue length at a particuar input port over 5×10^7 time slots. The on-line measuring period is chosen to be 1000 slots. We also let $\alpha(n) = \frac{1}{n}$ to obtain the sample mean estimates for the rates. In Figure 4, we compare our simulation results with those from the FIFO input buffered switch and the output buffered switch. As the FIFO input buffered switch suffers from the HOL blocking, its mean queue length blows up when ρ approaches 58% (see e.g., [26]). Our simulation results also show that our algorithm indeed achieves 100% throughput for the best-effort services, and its mean queue length is only moderately larger than that of the output buffered switch.

We note that 100% throughput can also be achieved for input buffered crossbar switch by using some weighted matching scheduling algorithm for each time slot (see e.g., [21], [22]). The computational complexity for such an algorithm is usually very high ($O(N^{2.5})$ per time slot). As our computational complexity is only $O(\log N)$ per time slot, it is much easier to implement. In comparison with the algorithms in [21], [22], theirs is "greedy at all time" and ours is "good on average." As perfect matching is not likely for all time, it is quite intuitive why a "good on average" policy like ours achieves 100% throughput with a much lower computational complexity per time slot. Moreover, it is easier to achieve desired *fairness* in our setting as the resource sharing problem can be formulated as an optimization problem for the "average" rates.

IV. Hardware implementation of the switch fabric

In this section, we report our implementation of the run time algorithm (Algorithm 3) for a 4×4 Birkhoff-von

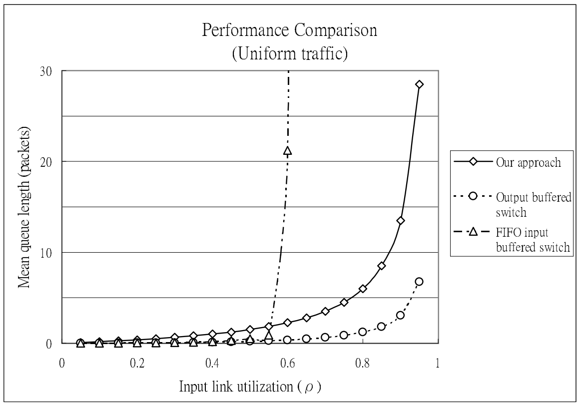

Fig. 4. Comparison of the mean queue length of three kinds of scheduling algorithms in a 4×4 crossbar switch

Neumann crossbar switch on a FPGA chip (Alter FPGA type : EPF10K100 ARC240-3). In Figure 5, we show the block diagram of the switch fabric. It consists of three main blocks: the data block, the controller block, and the crossbar block. All blocks inside the switch fabric are required to run on a common reference clock. Also, time is divided into slots and only a fixed sized packet is transferred in our switch during a time slot. The entire logic blocks were coded by high level behavioral language AHDL and synthesized by logic-synthesis tool MAX+Plus II.

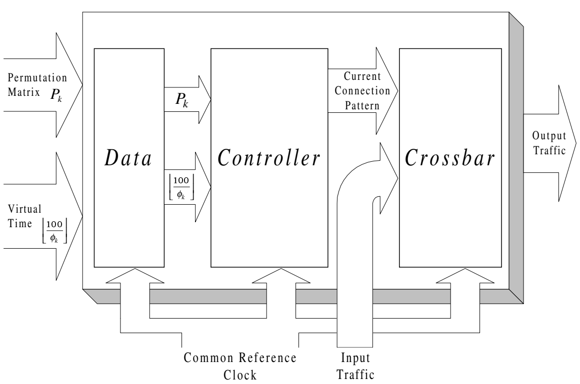

Fig. 5. The block diagram of Birkhoff-von Neumann crossbar switch

The main function of the data block is to load the parameters ϕ_i's and the corresponding permutation matrices P_i's into the controller block. Recall that ϕ_i's and P_i's need to be computed on-line within a measuring period as described in Section III-B. The controller block then uses these data to run Algorithm 3. Basically, it consists of a set of counters for computing the virtual finishing times and a comparator for determining the smallest virtual finishing time. For each time slot, the controller block determines an appropriate permutation matrix which is then used by the crossbar block to set up the corresponding connection pattern.

As we can only represent a real number with a finite number of bits in a digital system, we have to be careful of possible numerical errors in calculating virtual finishing times. In our design, the granularity of ϕ_i is chosen to be 0.01. This implies that the connection patterns by Algorithm 3 are repeated for every 100 time slots. In every 100 time slots, the number of slots that are assigned to the permutation matrix P_i should be $100\phi_i$. In view of the periodicity, all the virtual finishing times are reset to 0 every 100 time slots. Also, the increment of virtual finishing time $\frac{1}{\phi_i}$ is represented by the integer $\lfloor \frac{100}{\phi_i} \rfloor$, where the floor function $\lfloor x \rfloor$ represents the largest integer that is not larger than x. Such a representation ensures rate guarantees as one can easily verify that

$$\left\lfloor \frac{10000}{\left\lfloor \frac{100}{\phi_i} \right\rfloor} \right\rfloor = 100\phi_i \tag{20}$$

for any ϕ_i with the granularity 0.01. As the virtual finishing times are reset every 100 times slots, the maximum count for the virtual finishing time is $100\phi_i \lfloor \frac{100}{\phi_i} \rfloor \leq 10000$. This shows that a 14-bit latch is enough for each counter.

We note that choosing a granularity yields a sequence of periodic connection patterns. However, this is not the same as the frame structure used in the WRR algorithm [11], [17]. In our algorithm, the controller needs not to memorize the whole sequence of the connection patterns in the frame no matter how small the granularity is. It only needs to keep K matrices and thus reduces the design complexity of the controller.

To test the switch fabric, we use another Altera FPGA (type EPM7160ELC84-20) to generate the input traffic. This FPGA continues feeding a fixed traffic pattern to each input port of our crossbar switch so that we can verify the chip performance with a logic analyzer. In this test, we use the decomposition in (13). Clearly, we have $\phi_1 = 0.4$, $\phi_2 = 0.4$ and $\phi_3 = 0.2$, and the connection patterns are periodic with period 5. Figure 6 shows the verification from a logic analyzer. As shown in Figure 6, the four output ports (O1-O4) indeed have periodic outputs with period 5. Also, note that the counter is reset every 100 slots. Figure 7 shows the picture of the switch and source chip set. Our test also shows this relatively cheap FPGA chip can be run at the switching speed 14M slots/sec. With a 4-bit data bus, each input port can be run at the speed of 200M bits/sec. We believe the performance can be further improved by using full-customized ICs.

V. Multistage Birkhoff-von Neumann switches

As we have shown in Section II, the number of permutation matrices that need to be stored in an $N \times N$ Birkhoff-von Neumann switch is $O(N^2)$. This may not be scalable for a large switch. We resolve such a scaling problem by using multistage connecting networks. In

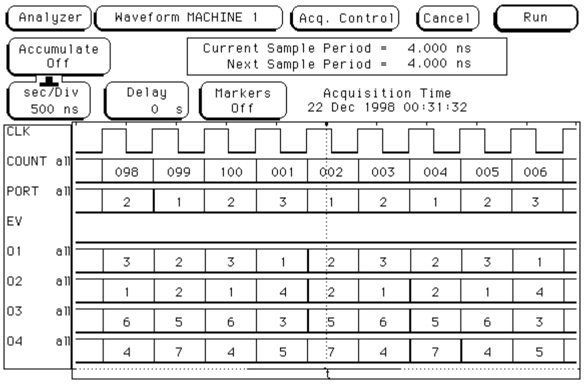

Fig. 6. The timing diagram of the switch fabric

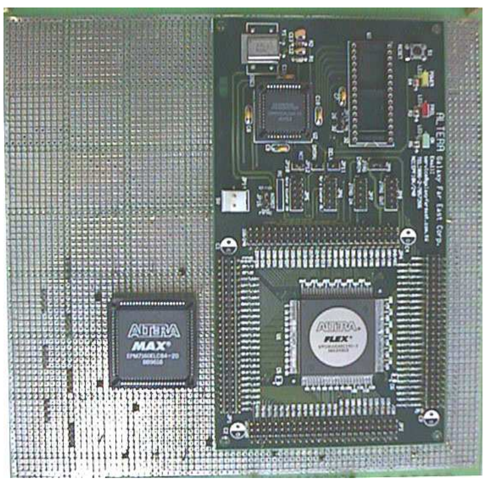

Fig. 7. The picture of chip set and demo board

this section, we propose using two types of connecting networks: a two-stage Banyan network and a three-stage rearrangeable network. For the ease of presentation, we only illustrate the idea by building 16×16 switches from 4×4 switches.

In Figure 8, we depict a two-stage Banyan network from 4×4 Birkhoff-von Neumann crossbar switches. The Banyan network is a 16×16 switch. At each stage, there are four 4×4 Birkhoff-von Neumann crossbar switches. The four output ports of a switch at the first stage are connected to the four switches at the second stage. By so doing, there is a unique path between each input-output pair. Also, as these switches are input buffered, there is no need for them to run synchronously.

To find the achievable rate region (or commonly known as the stability region in the queueing theory), we number the sixteen input ports by two dimensional indexing (i, j), where $i = 1, 2, 3, 4$, represents the index of the 4×4 switch at the first stage and $j = 1, 2, 3, 4$, represents the index of the port number of that 4×4 switch. Index the sixteen output ports similarly. Though not used in this paper, we note that such indexing allows self-routing. Let $r_{(i,j),(k,\ell)}$ be the rate assigned to the traffic from input (i, j) to output (k, ℓ). Since the "no overbooking" conditions for each 4×4 Birkhoff-von Neumann crossbar switch imply that the rate at each link in Figure 8 cannot be larger than 1, we have the following "no overbooking" conditions for the 16×16 Banyan network

$$\sum_{i=1}^{4} \sum_{j=1}^{4} r_{(i,j),(k,\ell)} \leq 1, \quad k, \ell = 1, 2, 3, 4, \tag{21}$$

$$\sum_{k=1}^{4} \sum_{\ell=1}^{4} r_{(i,j),(k,\ell)} \leq 1, \quad i, j = 1, 2, 3, 4, \tag{22}$$

$$\sum_{j=1}^{4} \sum_{\ell=1}^{4} r_{(i,j),(k,\ell)} \leq 1, \quad i, k = 1, 2, 3, 4. \tag{23}$$

Note that the "no overbooking" conditions for a 16×16 Birkhoff-von Neumann crossbar switch only require (21) and (22). Thus, the achievable rate region for such a Banyan network is much smaller than that of a 16×16 Birkhoff-von Neumann crossbar switch.

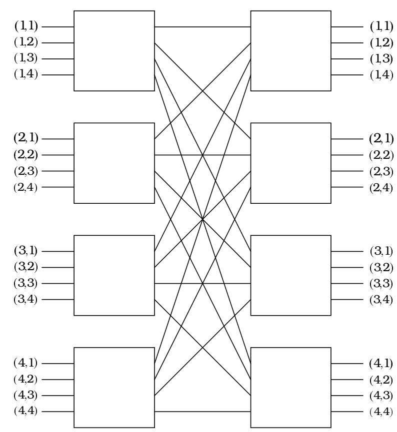

Fig. 8. A two-stage Banyan network from 4×4 Birkhoff-von Neumann crossbar switches

To eliminate the additional conditions in (23), one may add another stage as shown in Figure 9. Such a connecting network is a three-stage rearrangeable network (see e.g., [10]). The physical meaning of a rearrangeable network can be easily understood in the circuit switching setting. A call from an input port to an idle output port can always find a path by rearranging the paths of existing calls (the Slepian-Duguid algorithm). This implies that all the input ports can be connected to all the output ports at the same time as long as the input ports are all calling distinct output ports. Mathematically, a 16×16 permutation matrix can be represented by twelve 4×4 permutation matrices that correspond to the twelve 4×4 Birkhoff-von Neumann crossbar switches in the three-stage rearrangeable network. This is the reason why the achievable rate region of the three-stage rearrangeable network is the same as that of a 16×16 Birkhoff-von Neumann crossbar switch.

Now we show how one runs the PGPS algorithms in these twelve 4×4 Birkhoff-von Neumann crossbar switches.

Algorithm 4:
(i) As discussed in Section III, one first obtains the allocated rate matrix R for the 16×16 switch by on-line measurement and solving the corresponding optimization problem.
(ii) Use the Birkhoff-von Neumann decomposition algorithm described in Section II-A to find a set of positive numbers ϕ_k and 16×16 permutation matrices P_k, $k = 1, \ldots, K$ for some $K \leq 226$ that satisfies

$$R \leq \sum_{k=1}^{K} \phi_k P_k, \quad \text{and} \tag{24}$$

$$\sum_{k=1}^{K} \phi_k = 1. \tag{25}$$

(iii) For each 16×16 permutation matrix P_k, $k = 1, \ldots, K$, use the Slepian-Duguid algorithm to find the corresponding 4×4 permutation matrices $P_{k,j}$, $j = 1, 2, \ldots, 12$ (an alternative and faster algorithm was proposed by Lee, Hwang and Carpinelli [16]).
(iv) Let $R_j = \sum_{k=1}^{K} \phi_k P_{k,j}$, $j = 1, \ldots, 12$. Then R_j is the allocated rate matrix for the j^{th} 4×4 switch in the rearrangeable network. Note from (25) that R_j is a convex combination of 4×4 permutation matrices and thus a 4×4 doubly stochastic matrix.
(v) For each $j = 1, \ldots, 12$, use the Birkhoff-von Neumann decomposition algorithm described in Section II-A to find a set of positive numbers $\phi_{j,k}$ and 4×4 permutation matrices $\tilde{P}_{j,k}$, $k = 1, \ldots, K_j$ for some $K_j \leq 10$ that satisfies

$$R_j = \sum_{k=1}^{K_j} \phi_{j,k} \tilde{P}_{j,k}, \quad \text{and} \tag{26}$$

$$\sum_{k=1}^{K_j} \phi_{j,k} = 1. \tag{27}$$

The set of parameters $\phi_{j,k}$ and the corresponding permutation matrix $\tilde{P}_{j,k}$ are then used to run the PGPS scheduling algorithm in the j^{th} 4×4 switch.

Note that there are four different paths between each input-output pair of the three-stage rearrangeable network (as there are four choices of the intermediate stage). This requires additional book-keeping to split the traffic between each input-output pair. Let $r_{m,n,\ell}$, $m, n = 1, \ldots, 16$, $\ell = 5, 6, 7, 8$, be the allocated rate of the traffic from the m^{th} input port to the n^{th} output port by using the ℓ^{th} intermediate stage. Such rates can be computed in step (iii) of Algorithm 4. If the element at the (m, n) position of P_k is 1 and the corresponding matrices choose the ℓ^{th} intermediate stage, add ϕ_k to $r_{m,n,\ell}$. Once such rates are computed, the traffic from the m^{th} input port to the n^{th} output port via the ℓ^{th} intermediate stage should be splitted with the ratio $r_{m,n,\ell} / \sum_{\ell'=5}^{8} r_{m,n,\ell'}$.

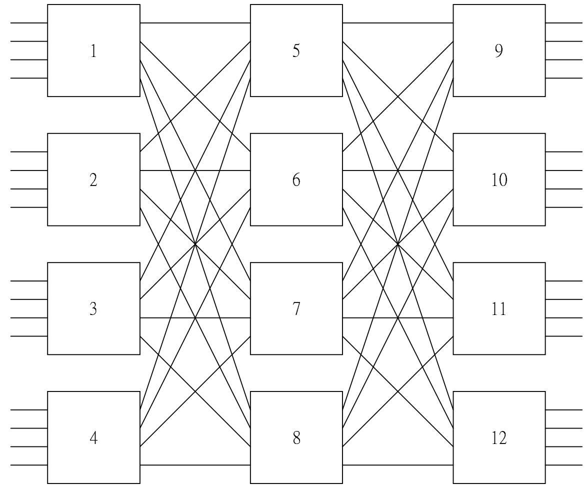

Fig. 9. A three-stage rearrangeable network of Birkhoff-von Neumann crossbar switches

VI. Conclusions

In this paper, we reported how one provides best-effort services in the Birkhoff-von Neumann switch. The problem is formulated as a resource sharing problem that can be optimized with respect to efficiency and fairness. An on-line measurement algorithm is proposed to measure the rate demands. Our simulation results show that our approach indeed achieves 100% throughput and that its mean queue length is only moderately larger than that of the output buffered switch.

We also reported our experience in designing the switch fabric of the Birkhoff-von Neumann switch. We showed that the line speed of 200 M bits/sec can be easily achieved by a FPGA chip and we expect that the performance can be further improved by using full-customized ICs.

As framing is not required for the Birkhoff-von Neumann switch, it is more flexible to connect them to form multistage networks. Two type of connecting networks were addressed: a two-stage Banyan network and a three-stage rearrangeable network. We showed that the two-stage Banyan network has a smaller achievable rate region than that of the Birkhoff-von Neumann switch with the same number of input-output ports. This problem can be fixed by adding another stage to form a three-stage rearrangeable network. To run the PGPS scheduling algorithms for the Birkhoff-von Neumann switches in a three-stage rearrangeable network, we proposed an algorithm for computing the rate decomposition.

There are several topics that require further investigation:

(i) Our on-line measurement only uses the information of the input processes. It can be further improved by collecting the information of the number of packets stored in the queues. However, this not only adds additional work for measurement, but also increases the complexity of solving the optimization problem.

(ii) The performance of multistage Birkhoff-von Neumann switching networks needs further study. It would be interesting to know the buffer requirements for the intermediate stage and the last stage. Also, one might study whether if there is a need for per-flow scheduling in a multistage Birkhoff-von Neumann switching network.

References

[1] T. Anderson, S. Owicki, J. Saxes and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems*, Vol. 11, pp. 319-352, 1993.

[2] C. Berge, *The Theory of Graphs and Its Applications* (translated by A. Doig). New York: Wiely, 1962.

[3] D. Bertsekas and R. Gallager, *Data Networks.* Prentice Hall, 1992.

[4] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucumán Rev. Ser. A*, Vol. 5, pp. 147-151, 1946.

[5] C.S. Chang, W.J. Chen and H.Y. Huang, "On service guarantees for input buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann," *IEEE IWQoS'99*, pp. 79-86, London, U.K., 1999.

[6] A. Charny, P. Krishna, N. Patel and R. Simcoe, "Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup," *IEEE IWQoS'98*, pp. 235-244, Napa, California, 1998.

[7] S.-T. Chuang, A. Goel, N. McKeown and B. Prabhkar, "Matching output queueing with a combined input output queued switch," *IEEE INFOCOM'99*, pp. 1169-1178, New York, 1999. 1997. Available from *http://tiny-tera.stanford.edu/~nickm/papers.html.*

[8] A. Demers, S. Keshav, and S. Shenkar, "Analysis and simulation of a fair queueing algorithm," in *Proc. SIGCOMM'89*, pp. 1-12, Austin, TX, Sept. 1989.

[9] L. Dulmage and I. Halperin, "On a theorem of Frobenius-König and J. von Neumann's game of hide and seek, *Trans. Roy. Soc. Canada III(3)*, Vol. 49, pp. 23-29, 1955.

[10] J. Hui, *Switching and Traffic Theory for Integrated Broadband Networks.* Boston: Kluwer Academic Publishers, 1990.

[11] A. Hung, G. Kesidis and N. Mckeown, "ATM input-buffered switches with guaranteed-rate property," *Proc. IEEE ISCC'98*, Athens, pp. 331-335, 1998.

[12] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, Vol. 27, pp. 1449-1455, 1979.

[13] F. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, Vol. 8, pp. 33-37, 1997.

[14] F. Kelly, A. Maulloo and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of Operations Research Society*, Vol. 49, pp. 237-252, 1998.

[15] P. Krishna, N.S. Patel, A. Charny and R. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE IWQoS'98*, pp. 225-234, Napa, California, 1998.

[16] H.Y. Lee, F.K. Huang and J.D. Carpinelli, "A new decomposition algorithm for rearrangeable Clos interconnection networks," *IEEE Trans. Commun.*, Vol. 44, pp. 1572-1578, 1996.

[17] T.T. Lee and C.H. Lam, "Path switching-a quasi-static routing scheme for large scale ATM packet switches," *IEEE Journal on Selected Areas of Communications*, Vol. 15, pp. 914-924, 1997.

[18] S. Li and N. Ansari, "Input-queued switching with QoS guarantees," *IEEE INFOCOM'99*, pp. 1152-1159, New York, 1999.

[19] A.W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and Its Applications.* New York: Academic Press, 1979.

[20] L. Massoulié and J. Roberts, "Bandwidth sharing: objectives and algorithms," *IEEE INFOCOM'99*, pp. 1395-1403, New York, 1999.

[21] N. McKeown, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Proc. IEEE INFOCOM'96*, pp. 296-302, 1996.

[22] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *Proc. IEEE INFOCOM'98.*

[23] L. Mirsty, *Transversal Theory.* New York: Academic Press, 1971.

[24] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithm and Complexity.* New Jersey: Prentice-Hall, 1982.

[25] A.K. Parekh and R.G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the single-node case," *IEEE/ACM Transactions on Networking*, Vol. 1, pp. 344-357, 1993.

[26] M. Schwartz, *Broadband Integrated Networks.* New Jersey: Prentice Hall, 1996.

[27] D. Stiliadis and A. Varma, "Providing bandwidth guarantees in an input-buffered crossbar switch," *Proc. IEEE INFOCOM'95*, pp. 960-968, 1995.

[28] I. Stoica and H. Zhang, "Exact emulation of an output queueing switch by a combined input output queueing switch," *IEEE IWQoS'98*, pp. 218-224, Napa, California, 1998.

[29] J. von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem, " *Contributions to the Theory of Games,* Vol. 2, pp. 5-12, Princeton University Press, Princeton, New Jersey, 1953.